

A BYTE BOOK

>> \$9.95

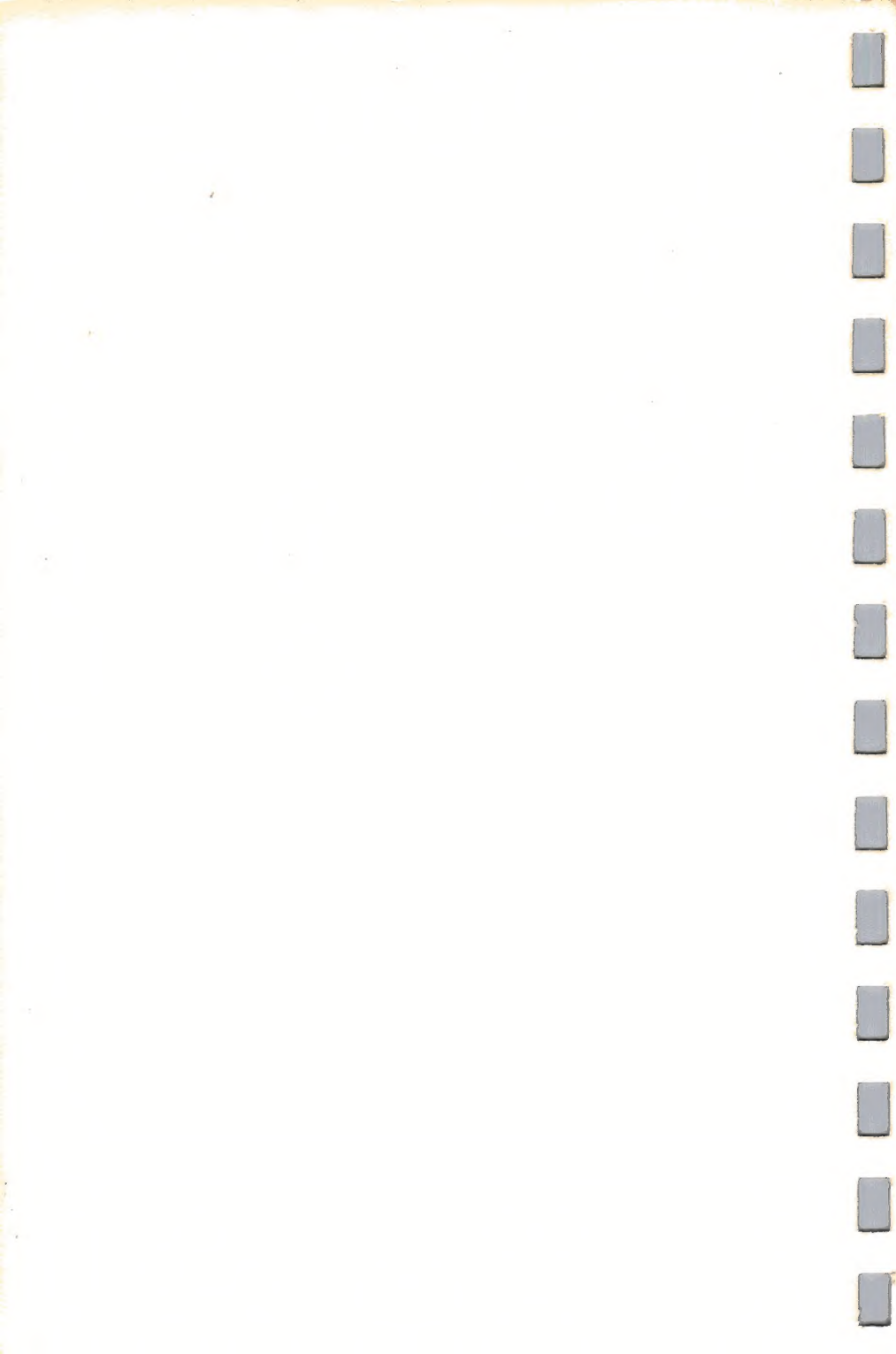
050N 134  
PRICE 9.95  
M 3H 0190

# GOSUBS

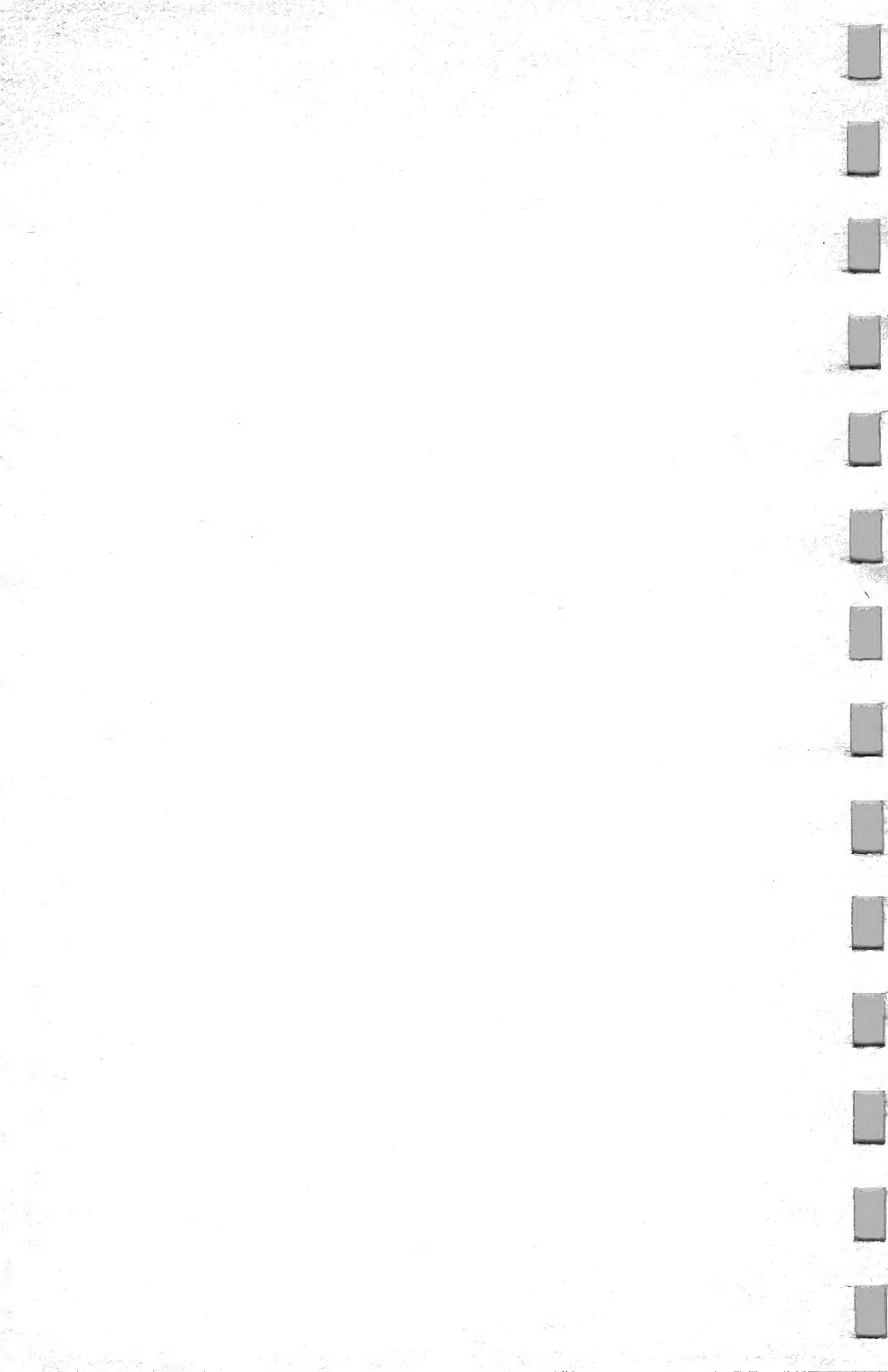
## 100

Program-Building Subroutines  
in  
**Timex/Sinclair BASIC**

Ewin Gaby and Shirley Gaby



GOSUBS



# GOSUBS

---

**100**

**Program-Building Subroutines  
in  
Timex/Sinclair BASIC**

---

Ewin Gaby and Shirley Gaby

McGRAW-HILL BOOK COMPANY

New York St. Louis San Francisco Auckland Bogotá  
Guatemala Hamburg Johannesburg Lisbon London Madrid  
Mexico Montreal New Delhi Panama Paris San Juan  
São Paulo Singapore Sydney Tokyo Toronto

The author of the programs provided with this book has carefully reviewed them to ensure their performance in accordance with the specifications described in the book. Neither the authors nor McGraw-Hill, however, makes any warranties whatever concerning the programs. They assume no responsibility or liability of any kind for errors in the programs or for the consequences of any such errors.

As used in this book, the terms "Timex/Sinclair 1000" and "T/S 1000" refer to the Timex/Sinclair 1000, a computer manufactured and sold by the Timex Computer Corporation. Timex/Sinclair 1000 is a registered trademark of the Timex Corporation. Timex/Sinclair 1500 and Timex/Sinclair 2000 are also registered trademarks of the Timex Corporation.

GOSUBS: 100 Program-Building Subroutines in Timex/Sinclair BASIC

Copyright © 1984 by Ewin Gaby and Shirley Gaby. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

A BYTE Book.

1 2 3 4 5 6 7 8 9 0      S E M S E M      8 9 3 2 1 0 9 8 7 6 5 4 3

**ISBN 0-07-022677-6**

LIBRARY OF CONGRESS CATALOGING IN PUBLICATION DATA

Gaby, Ewin.

GOSUBS : 100 program-building subroutines in Timex/  
Sinclair BASIC.

Includes index.

1. Timex 1000 (Computer)—Programming. 2. Basic  
(Computer program language) I. Gaby, Shirley. II. Title.  
QA76.8.T48G23 1984 001.64'2 83-16220  
ISBN 0-07-022677-6

The editors for this book were John A. Aliano and Paul Farrell;  
and the editing supervisor was Charles P. Ray.

# Contents

Preface		vii
Chapter 1	How to Use this Book	1
2	Introduction to Subroutines	6
3	Area and Volume	18
4	Conversion	29
5	Statistics	51
6	Business	66
7	Maximum, Minimum, and Sequence	85
8	<b>INKEY\$</b> and <b>SCROLL</b>	99
9	Graphing	112
10	Tables	127
Index		153





# Preface

Books about the Timex/Sinclair Computer are becoming more and more common. Yet, a great majority of them seem to address only games or the intricate generalities of "programming."

Now, here is a book that addresses specific needs of the home, business, and school; describes how to assemble programs in cookbook fashion; and provides a large number of tested subprograms which can be used straight from the book.

With this book, your programming skills are less important than your ability to determine what you want to accomplish! What's more, your skills will grow as you use it. The content is arranged to lead you from simple to more complex concepts, and to support you all the way. By using the building blocks provided, and the many helps and suggestions which accompany them, you'll soon be able to construct many useful programs.

Combining your own creative ideas with this book's concepts, you'll quickly develop better and more unique programs. *You* become the powerful force behind your computer.

Using the building blocks furnished, it is possible to design programs for checkbook registers, budgets, accounts payable, ledger sheets, statistics, test scores, percentile ranking, inventory, tax returns, directories and much more. In addition, the many conversion, mathematics, plotting, and program control subroutines will enrich your ability to write unique and exciting programs.

If you want to use your computer for more than games, then this book is for you. With it, you can easily learn to design quality programs that will do all of those useful things you thought a computer should do—and more!

Ewin Gaby and Shirley Gaby

GOSUBS



# Chapter 1

## How to Use this Book

- The equipment
- The audience
- Typographical conventions used in this book
- Using this book with the T/S 2000 and Spectrum
- Suggestions: keywords and **SAVE**

### The Equipment

The programs in this book are designed to run on the Timex/Sinclair 1000, the Sinclair ZX81, and the Sinclair ZX80 (with 8K ROM). With a very few exceptions, these programs also will run on the Timex/Sinclair 2000 and the Sinclair Spectrum color computers. (Even the few exceptions will run on the 2000 and the Spectrum with only minor modifications.)

### The Audience

The book is designed to provide both the novice and the experienced programmer with useful “building blocks” from which programs can be constructed, modified, and refined. These building blocks are parts of programs which perform specific tasks and they are called *subroutines*.

The subroutines are arranged in a sequence that should make it easy for the novice to proceed from simple, easily understood computer concepts toward fairly complex ones. And while this arrangement will give the novice a better understanding of the operation of the computer, it will also provide the experienced programmer with a wide range of ready-to-use subroutines.

Each chapter begins with a careful description of the subroutines under discussion, and makes suggestions about how the subroutines may be used. In some cases, illustrative sample programs demonstrate how subroutines can be combined.

In order to use the subroutines in this book, it is not necessary to read the text. However, the text will provide the reader with useful information about the design, use, and possible modification of all the subroutines. For the novice, the text should be considered a self-teaching guide, starting with the simplest applications and leading to complex computer functions, including an introduction to machine language programs.

### **Typographical Conventions Used in This Book**

The conventions used throughout the book are relatively standard, and will for the most part be familiar to those who have used the Timex/Sinclair and Sinclair Manuals. All keyword commands and all functions are printed in **BOLDFACE**. Consider the following program line:

```
20 LET Z=SQR(X*Y)
```

In this line, **LET** is a keyword command and **SQR** is a function, and each is obtained with a single keystroke.

A single space is represented in program lines by an underscore (\_). Care should be taken to key in the proper number of spaces within quotes. For example, consider the following two lines:

```
20 LET A$=""
```

and

```
20 LET A$=" _"
```

In the first example, there is no space between the two quote marks. In the second example, there is a space.

Occasionally, when a long line of similar characters is required, the number of characters will be indicated. Thus, if we want you to put the following line on the computer's screen:

```
20 LET A$="....."
```

we would indicate it as follows in this book:

```
20 LET A$="...[25 colons]..."
```

In order to indicate an inverse character we will show the character much as it will appear on the screen:

**B**

This character is the *inverse* version of the letter B. It is obtained by first putting the computer in the graphics mode (shifted-9) and then pressing the B key.

### Using This Book with the T/S 2000 and Spectrum

The Timex/Sinclair 2000 and Sinclair Spectrum color computers have many facilities not found in the Timex/Sinclair 1000 or ZX Computers. With a few exceptions (noted in later chapters), these added facilities neither facilitate nor inhibit the use of the subroutines in this book.

If you do have one of the color computers, however, you may wish to modify some of the subroutines to take advantage of your computer's special features. One obvious feature available on the color computers is the ability to compact a number of program lines into a single program line. You may wish to apply this feature to some of the subroutines. Two examples of this feature are:

[T/S 1000, ZX]

10 **LET X=Ø**

20 **LET Y=Ø**

30 **LET Z=Ø**

40 **[etc.]**

[T/S 2000, Spectrum]

10 **LET X=Ø: LET**

**Y=Ø: LET Z=Ø**

20 **[etc.]**

and

10 **FOR I=1 TO 10**

20 **PRINT I**

30 **NEXT I**

40 **STOP**

10 **FOR I=1 TO 10:**

**PRINT I: NEXT I**

20 **STOP**

If you have a T/S 2000 or a Sinclair Spectrum, it is *not*

necessary for you to make these changes. If you choose to compact a program with your color computer, take care to adjust the line numbers in any **GOTO** statements so that the compacted program will operate properly.

### Suggestions: Keywords and **SAVE**

Although the following suggestions are not confined to subroutines, they will be helpful to you in programming the Timex/Sinclair computer.

Many of the subroutine titles contain words which are the same as the computer's keywords. When this occurs, memory space and typing time will be saved by simply using the keyword instead of typing in the letters one by one. For instance:

```
20  REM INPUT X AND SAVE
```

All the characters except the line number and X are keyword commands. The method for typing keywords is simple, once it is understood. In the example above, type 2 and 0, then E. (Because the computer is looking for a keyword after the 20, the E will be interpreted as **REM**.) Now type **THEN** (shifted-3), and notice that the cursor becomes a **K**, which indicates that the computer is expecting a keyword following **THEN**. Now, press the I key, and **INPUT** will appear on the screen. To remove the **THEN**, backspace (shifted-4), delete (shifted-0), and forespace (shifted-8). Now, type X and type **AND** (shifted-2). At this point we are ready to type the **SAVE**, using the same method as was used for typing **INPUT**: Type **THEN** (shifted-3), type **SAVE** (on the S key), and then backspace, delete, and forespace.

Anytime that you want to use any keyword in a **REM** statement or within quotations in a **PRINT** statement, you can type **THEN**, type the keyword, backspace, delete, and forespace. Doing this is usually easier than typing the word letter-by-letter, and will also conserve memory.

We also have a suggestion about how to **SAVE** a program. Your computer manual describes how a program can be saved on tape by using a tape recorder. If your program contains stored data, then that data, too, will be saved. However, if



the program is **RUN** after **LOADing**, the data will be cleared. In order to use the program *and* the stored data, you must **GOTO** the first line of the program, which does not initialize your data.

A convenient way to **SAVE** your program and data is to make the last two lines of your program read as follows:

```
9998  SAVE "PROGRAM"
```

```
9999  GOTO 2000
```

On line 9998, we have placed **SAVE**, followed by the name of the program in quotations. On line 9999, we have told the computer to **GOTO 2000** (or whatever line number is needed to start the program).

When you are ready to **SAVE** the program and data, just start your recorder and then tell the computer **GOTO 9998**. Doing this will **SAVE** the program in the normal manner. When you **LOAD** the saved program back into the computer, the program will go to line 2000 as soon as it is loaded. Thus, if line 2000 leads to a printing routine, your printed data will "pop" onto the screen as soon as the program is loaded.

## Conclusion

Although some of these subroutines are simple operations, many of them contain interesting and complex ideas. If you are interested in learning more about your computer, then analyzing these subroutines will provide you with new ideas and directions for your programming efforts. In Chapter 2, we'll introduce the fundamental rules by which subroutines operate. In Chapter 3, our first group of subroutines is introduced and explained.

# Chapter 2

## Introduction to Subroutines

- A definition, with examples
- Numbering subroutine lines
- Subroutine formats
- Variables

### A Definition, with Examples

Let's begin with a definition: A *subroutine* is a section of a computer program that is called into execution by a **GOSUB** command, and is terminated with a **RETURN** command.

The **GOSUB** command, just like **GOTO**, causes a computer to "jump" to a specified program line. However, the **GOSUB** also causes the computer to store in its memory the line number *from which it jumped*. This stored line number allows the computer to come back to that line after the subroutine is completed.

As the computer moves through the program, it encounters a **GOSUB** command. The computer immediately jumps to the line specified by the **GOSUB**, and continues to progress through the lines of the subroutine until it comes to a **RETURN** command. The computer then returns to the body of the program exactly one line after the line containing the **GOSUB** instruction. (The **RETURN** line is part of the subroutine; the **GOSUB** line is not.)

That is all there is to a subroutine. It is a segment of a program to which the computer will jump as the result of a **GOSUB** command, and from which the computer will jump back when it encounters the **RETURN** command. Here is an example:

```
10 LET A=0
20 FOR I=1 TO 10
```

```

30  GOSUB 200
40  PRINT I;" - ";A
50  NEXT I
60  STOP
200  REM SUBROUTINE
210  LET B=A+I
220  LET A=(A+B)*3
230  RETURN

```

Each time the computer executes line 30, it is told to **GOSUB 200**, and so it jumps to line 200, where it continues its step-by-step execution. When the computer reaches the **RETURN** command at line 230, it jumps back to line 40, which is the line following the last line executed in the main program, and then continues its step-by-step execution.

According to the definition given earlier, a single line containing only the **RETURN** command would qualify as a subroutine:

```

100  GOSUB 300
.
.
.
300  RETURN

```

This example is not a *useful* subroutine, however, since the program simply jumps from line 100 to line 300 and immediately returns.

A two-line subroutine, by contrast, *can* be made to perform a useful task:

```

100  GOSUB 400
.
.
.
400  LET X=3*(Y-5)
410  RETURN

```

The problem is, why use a subroutine to accomplish what can be accomplished in a single program line? What is the point of sending the computer to search for, find, and process the single line in a subroutine, and then return to the previous program? The answer depends upon your personal preference.

Compare Programs A and B:

[Program A]  
(with subroutine)

```
100  GOSUB 400
.
.
.
200  GOSUB 400
.
.
.
300  IF Y>5 THEN
.    GOSUB 400
.
.
390  STOP
400  REM SAMPLE
      SUBROUTINE
410  LET X=3*(Y-5)
420  RETURN
```

[Program B]  
(without subroutine)

```
100  LET X=3*(Y-5)
.
.
.
200  LET X=3*(Y-5)
.
.
.
300  IF Y>5 THEN LET
      X=3*(Y-5)
.
.
.
390  STOP
```

#### Observation 1

A two-line subroutine may be useful as a convenience, but is not likely to be an efficient use of computer time and memory space.

While we are on the subject of efficient use of time and space, we might make another observation:

#### Observation 2

Any subroutine that is called only once by a program is better placed within the body of the program.

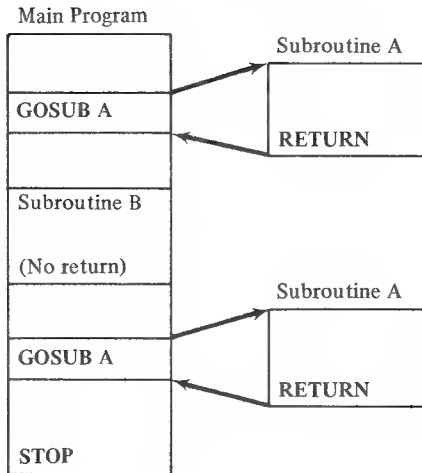


Figure 2-1

If you wish to use a subroutine from this book, but need to use it only once in your program, then you can simply insert the subroutine in the proper place in your program and remove the **RETURN** command.

Figure 2-1 depicts the single use of Subroutine B and, in contrast, the multiple use of Subroutine A.

One subroutine can call for another subroutine. This “nesting” of subroutines can sometimes simplify the writing of programs, but makes the program difficult to follow. An example of this is shown in Figure 2-2, where Subroutines A, B, and C are each called only once by the main program. However, subroutines A and B appear a number of times, since they are called by the other subroutines.

Subroutines can be called conditionally. By this we mean that a subroutine will be called only if a certain condition exists. Consider the following statements:

```
10 IF A>B THEN GOSUB 200
```

```
10 IF X>0 AND X<100 THEN GOSUB 300
```

In the first line shown above, the program will **GOSUB** to line 200 only if A is greater than B. The second line requires that X must lie between 0 and 100 for the program to **GOSUB** 300.

Subroutines can also be called to a line designated by a variable, as follows:

```
10 LET NUM = INT(RND*3)+1
```

```
20 GOSUB NUM*100+1000
```

```
10 INPUT Z
```

```
20 GOSUB Z*1000
```

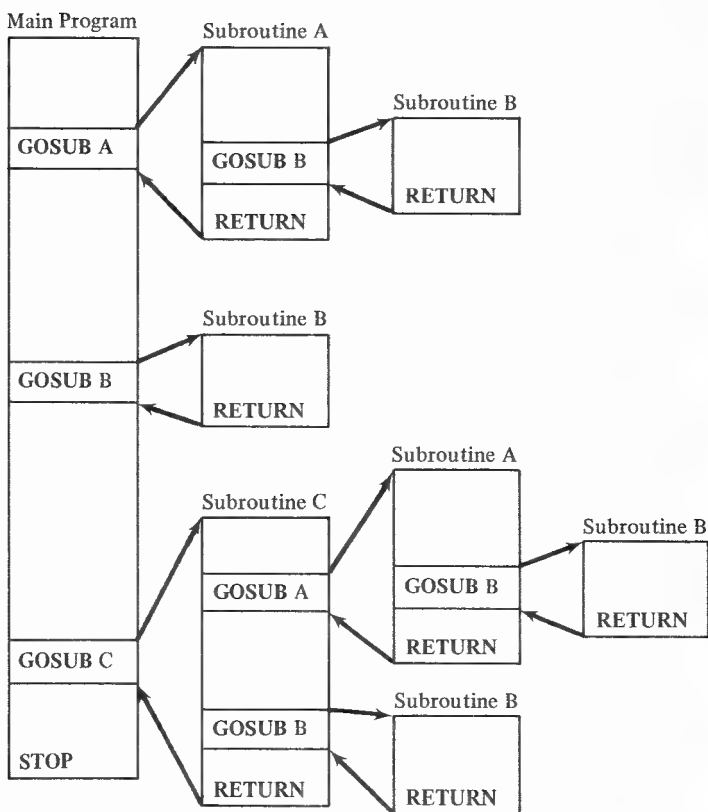


Figure 2-2

In the first pair of lines, the program goes to a subroutine at 1100, 1200, or 1300, depending upon the randomly generated value of NUM. In the second pair of lines, the program jumps to a subroutine determined by the value of Z, which is supplied by the operator.

### Numbering Subroutine Lines

The subroutines in this book have been designed to have a wide application. In order for these subroutines to work properly, however, they must be modified to match the numbering and the variables of your program.

The first and perhaps the most obvious modification you will have to make is to change the line numbers shown in the subroutines. You will notice that each line number is three characters long, with the first character being X. This form is used so that you can assign a line number to the subroutine by simply replacing the X with an appropriate digit. Consider the following subroutine:

```
X00  REM SAMPLE SUBROUTINE
X10  LET A=B
X20  RETURN
```

If you wanted to begin this subroutine at line 800, you'd simply replace the X with 8, as shown:

```
800  REM SAMPLE SUBROUTINE
810  LET A=B
820  RETURN
```

Any **GOTO** commands embedded in a subroutine in this book will also show up with an extra X, and will require your modification. Note the reference at the end of line X30:

```
X00  REM SAMPLE SUBROUTINE
X10  LET A=5
X20  LET R=R+A*5
X30  IF R<50 THEN GOTO X20
X40  RETURN
```

If you were to use this subroutine in one of your programs, you'd have to replace six, not five, X's.

When the command **GOSUB** is embedded in a subroutine, the first digit of the following number is indicated by the letter Y or Z. The Y or Z indicates that some digit other than X is required. The name of the required subroutine is then shown in brackets, as in line X40:

```

X00  REM SAMPLE SUBROUTINE
X10  LET A=5
X20  LET R=R+A*5
X30  IF R<50 THEN GOTO X20
X40  GOSUB Y00 [Feet to Meter Subroutine, 4.6a]
X50  RETURN

```

### Subroutine Formats

The subroutines in this book are constructed so that the name of each subroutine and its input and output variables are always listed in lines X00 to X09. The subroutine itself is always listed from lines X10 to X90. Thus, all subroutines begin at a line number which is a multiple of 100, and use no more than 90 line numbers in the subroutine.

This numbering method is very helpful to the programmer, since it allows subroutines to be assigned in sequential hundreds. For instance, you might assign the subroutine lines as follows: line 600 to the conversion subroutine, line 700 to the printing subroutine, and line 800 to the Z-Score subroutine.

Now you would simply use 6 to replace X in the *conversion* subroutine, 7 to replace X in the *printing* subroutine, and 8 to replace X in the *Z-Score* subroutine. (Line numbers such as: 2100, 3500, and 8800 could just as easily have been used, replacing the X with 21, 35, or 88, respectively.)

### Variables

There are three types of variables used in the subroutines in this book: input variables, working variables, and output variables.



*Input variables* are those variables which are passed to the subroutine when it is called by the main program, or by another subroutine.

*Working variables* are those variables which are used only within the subroutine. As such, they are a kind of temporary variable.

*Output variables* are those variables which are passed from the subroutine to the calling program when the computer returns from the subroutine. Output variables are the results of the subroutine's calculation or manipulation.

Consider the following example, where X is an input variable and is passed to the subroutine as a definite value, A is a working variable and is used only within the subroutine, Y is an output variable and passes back a definite value to the main program:

```

10  LET X=INT (RND*15)+1
20  GOSUB 300
.
.
.
300 REM SAMPLE SUBROUTINE
310 LET A=0
320 IF X<10 THEN LET A=5
330 LET Y=3+A*5
340 RETURN

```

Notice that if the value of X had been changed within the subroutine, then it would have been both an input variable and an output variable. We can illustrate this by replacing line 340 with:

```

340 LET X=X+A
350 RETURN

```

Now the input variable is X, the working variable is A, and the output variables are Y and X.

When you use any of the subroutines in this book, you must take care that the variables in your program match the

appropriate input and output variables of the subroutine. Matching the variables will usually require that the names of the variables be changed either in the subroutine or in the main program. Suppose, for example, that you are writing the following program:

```

.
.
.
50  LET A=3*X
60  GOSUB 300
.
.
.
290 STOP
300 REM INCHES TO FEET
301 REM INPUT IN
302 REM OUTPUT FT
310 LET FT=IN/12
320 RETURN

```

In this program, the variable to be passed to the subroutine from the main program is named A, but the subroutine input variable is named IN. One of these variables must be changed so that the variable passed to the subroutine and the input variable are the same. This can be done by changing line 50 in the program to:

```
50  LET IN=3*X
```

If we change line 50, then we must be careful to change *every* A in the entire program to IN.

Alternately, we can change line 310 in the subroutine to:

```
310 LET FT=A/12
```

In most cases, changing line 310 will be the best approach. However, in some complicated subroutines, this change may be difficult or impossible without much rewriting. In such a

case, the easiest method of changing the variables may be to simply insert a line like the following:

```
55  LET IN=A
```

If this conversion line is used, then no other changes related to the input variable are necessary. Output variables may be handled in the same manner, by converting the output variable back to a main program variable. This last method allows the use of multiple variables, as shown below:

```

:
:
50  LET A=3*X
55  LET IN=A           [convert A to input variable IN]
60  GOSUB 300          [convert IN to FT]
65  LET AF=FT          [convert output variable FT to AF]
:
:
110 INPUT B
115 LET IN=B           [convert B to input variable IN]
120 GOSUB 300          [convert IN to FT]
125 LET BF=FT          [convert output variable FT to BF]
:
:
160 LET C=15
165 LET IN=C           [convert C to input variable IN]
170 GOSUB 300          [convert IN to FT]
175 LET CF=FT          [convert output variable FT to CF]
:
:
290 STOP
300 REM INCHES TO FEET

```

```

301  REM INPUT IN
302  REM OUTPUT FT
310  LET FT=IN/12
320  RETURN

```

In the above program, AF, BF, and CF are the equivalent, in feet, of the variables A, B, and C, which are in inches.

### Observation 3

Program variables passed to subroutines must match the subroutine input variables, and subroutine output variables passed back to the program must match program variables.

### Observation 4

In some cases, in order to avoid confusion, program variables should be converted to subroutine input variables in the program line immediately preceding the **GOSUB** line. Likewise, subroutine output variables should be converted to program variables in the program line immediately following the **GOSUB** line.

Working variables may also require some changes. Although working variables are used only within a subroutine, they can affect your program. If the name of a working variable is the same as the name of one of your program variables, then the computer will not be able to distinguish between them. Under some circumstances, this confusion between variables will cause problems.

Because the computer searches for subroutine line numbers sequentially, beginning at line 001, programs will run slightly faster if the subroutines are placed *early* in the program. This can easily be done. You can assign your first 99 lines (lines 1 through 99) to be used for “initialization,” lines 100 through 2999 for subroutines, and lines 3000 and up for the program. This means that your program’s name and any dimension or presetting statements—such as **DIM A(4,20)** or **LET N=0**—can be placed in the first 98 lines. Line 99 will then be: **99 GOTO 3000**. The main program will be contained

in the lines beginning at 3000. All of your subroutines will then be placed between line 99 and line 3000. Such a scheme will leave space for up to 29 subroutines of 100 lines each, and they can all be placed near the front of the program.

As a final observation, it should be noted that the command **STOP** should be placed between the main program and any subroutines which follow the main program. Placing the **STOP** in this position will make certain that when the computer completes the main program, it will not continue merrily on through the subroutines which follow. (See the example on page 14, which includes the **STOP** command in line 290.)

#### Observation 5

Use a **STOP** command between the end of the main program and any subroutines which follow the main program.

The remainder of this book consists of listings of various subroutines which can be used in your programs. Used carefully, in the manner described in this chapter, subroutines will add flexibility to your programs with little effort on your part. As your confidence and understanding about subroutines grow, you will begin to modify them to meet your needs, and soon you will be writing your own subroutines.

## Chapter 3

# Area and Volume

The subroutines in this chapter and in Chapter 4 are all simple computations and conversions. Many of them contain only a single line of operating program. Still, they are useful subroutines, and can save you much time and effort in constructing your programs.

For many of the calculations in this chapter, more than one subroutine is given. If you encounter a number of subroutines that all solve the same problem, the type of input information you have will determine which subroutine you should use. For instance, if you wish to find the area of a square, you will want to use either Subroutine 3.5 or 3.6. Subroutine 3.5 requires the length of a side of the square as input variable, while Subroutine 3.6 requires the length of the square's diagonal.

Notice that in those subroutines dealing with circular measurements, the symbol **PI** is used. **PI** is obtained by pressing the M key when the computer is in the function or **F** mode. (Other trigonometric functions, such as **TAN** and **SIN** are likewise obtained only when the computer is in the **F** mode.) Putting the computer in **F** mode is accomplished by pressing the **SHIFT** and **ENTER** keys simultaneously. Doing this changes the cursor from **L** to **F**.

In the event that the input variables of a needed subroutine do not match your available data, you can modify the subroutine to accommodate your data. An example of such modifi-

cation can be seen in Subroutine 3.9, which calculates the area of a regular polygon. If your data includes the perimeter of the polygon but does not include the length of a side, then simply delete line X10 from the subroutine and change line X01 from:

```
X01  REM INPUT S AND N
```

to

```
X01  REM INPUT P AND N
```

This change in line X01 will serve as a reminder that the input variables of the subroutine have been changed.

Although the subroutines given in this chapter should meet most of your needs, you can easily design any additional subroutines which you may find useful. If you have a formula which is of the form:

$X =$  (Some formula for X stated in terms of Y)

then your subroutine is simply:

```
X10  LET X= (Some formula for X stated in terms of Y)
X20  RETURN
```

For example, if X is the volume of a sphere, and the formula for that volume is :

$$\frac{4}{3}\pi y^3$$

then the BASIC subroutine would look like this:

```
X10  LET X=4*PI*Y**3/3
X20  RETURN
```

You can apply a similar process to any formula to design a new subroutine.

## AREA OF A TRIANGLE

**PURPOSE:** Calculation

**3.1 NAME:** Area of Triangle, Given Base and Height.

**INPUT VARIABLES:**  $H$  = height;  $B$  = base.

**OUTPUT VARIABLES:**  $T$  = area of triangle.

X00 **REM** AREA OF TRIANGLE

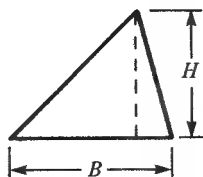
( $B, H$ )

X01 **REM** INPUT  $B$  AND  $H$

X02 **REM** OUTPUT  $T$

X10 **LET**  $T = B * H / 2$

X20 **RETURN**



**3.2 NAME:** Area of Triangle, Given Base and Two Angles.

**INPUT VARIABLES:**  $B$  = base;  $A_1$  = angle one;  $A_2$  = angle two.

**OUTPUT VARIABLES:**  $T$  = area of triangle;  $H$  = height.

X00 **REM** AREA OF TRIANGLE

( $B, \text{ANGLE}$ )

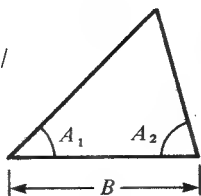
X01 **REM** INPUT  $B, A_1$  AND  $A_2$

X02 **REM** OUTPUT  $T$  AND  $H$

X10 **LET**  $H = \frac{\tan A_1 \cdot \tan A_2 \cdot B}{(\tan A_1 + \tan A_2)}$

X20 **LET**  $T = B * H / 2$

X30 **RETURN**



**3.3 NAME:** Area of Equilateral Triangle.

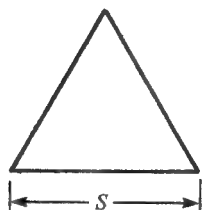
**INPUT VARIABLES:**  $S$  = length of any side.



OUTPUT VARIABLES: H = height; T = area of triangle.

```

X00  REM AREA OF
      EQUILATERAL TRI
X01  REM INPUT S
X02  REM OUTPUT H AND T
X10  LET H=S*SQR 0.75
X20  LET T=S*H/2
X30  RETURN
  
```



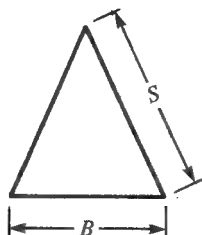
### 3.4 NAME: Area of Isosceles Triangle, Given One Side and the Base.

INPUT VARIABLES: B = base; S = side.

OUTPUT VARIABLES: H = height; T = area of triangle.

```

X00  REM AREA OF ISOSCELES
      TRI
X01  REM INPUT B AND S
X02  REM OUTPUT H AND T
X10  LET H=SQR(S*S-(B/2)
      **2)
X20  LET T=B*H/2
X30  RETURN
  
```



## AREA OF A SQUARE

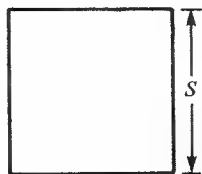
PURPOSE: Calculation

### 3.5 NAME: Area of a Square, Given One Side.

INPUT VARIABLES: S = length of one side.

OUTPUT VARIABLES:  $A$  = area of square.

```
X00 REM AREA OF SQ(SIDE)
X01 REM INPUT S
X02 REM OUTPUT A
X10 LET A=S*S
X20 RETURN
```

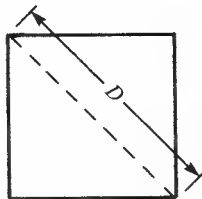


### 3.6 NAME: Area of Square, Given the Diagonal.

INPUT VARIABLES:  $D$  = length of the diagonal of the square.

OUTPUT VARIABLES:  $S$  = length of a side;  $A$  = area of square.

```
X00 REM AREA OF SQ(DIAG)
X01 REM INPUT D
X02 REM OUTPUT S AND A
X10 LET S=D/SQR2
X20 LET A=S*S
X30 RETURN
```



## AREA OF A RECTANGLE

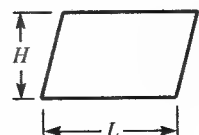
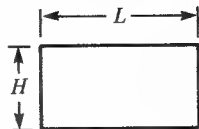
PURPOSE: Calculation

### 3.7 NAME: Area of Rectangle (or Parallelogram) Given Length and Height.

INPUT VARIABLES:  $L$  = length;  $H$  = height.

OUTPUT VARIABLES:  $R$  = area of rectangle.

```
X00 REM RECTANGLE AREA
      (L,H)
X01 REM INPUT L AND H
X02 REM OUTPUT R
X10 LET R=L*H
X20 RETURN
```



### 3.8 NAME: Area of Rectangle Given Diagonal and Height.

INPUT VARIABLES:  $H$  = height;  $D$  = diagonal.

OUTPUT VARIABLES:  $R$  = area of rectangle;  $L$  = length.

X00 **REM** RECTANGLE AREA  
(DIAGNL)

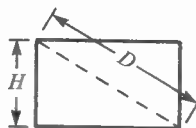
X01 **REM** INPUT  $H$  AND  $D$

X02 **REM** OUTPUT  $R$  AND  $L$

X10 **LET**  $L = \text{SQR}(D * D - H * H)$

X20 **LET**  $R = L * H$

X30 **RETURN**



## AREA OF A REGULAR POLYGON

**PURPOSE:** Calculation

### 3.9 NAME: Area of any Regular Polygon.

INPUT VARIABLES:  $S$  = length of a side;  $N$  = number of sides.

OUTPUT VARIABLES:  $A$  = area of polygon;  $H$  = height to center;  $P$  = perimeter of polygon.

X00 **REM** POLYGON AREA

X01 **REM** INPUT  $S$  AND  $N$

X02 **REM** OUTPUT  $A, H$ , AND  $P$

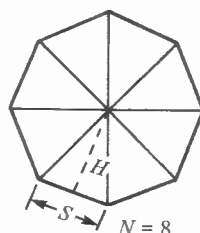
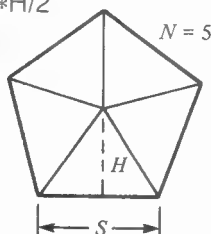
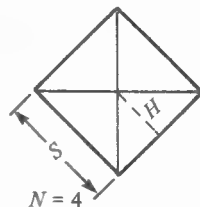
X10 **LET**  $P = S * N$

X20 **LET**  $T = 2 * \text{PI} / N$

X30 **LET**  $H = S / (2 * \text{TAN}(T/2))$

X40 **LET**  $A = P * H / 2$

X50 **RETURN**



## AREA OF A CIRCLE

**PURPOSE:** Calculation

**3.10 NAME:** Area of Circle, Given the Radius.

**INPUT VARIABLES:**  $R$  = radius of circle.

**OUTPUT VARIABLES:**  $C$  = area of circle.

X00 **REM** CIRCLE AREA

(RADIUS)

X01 **REM** INPUT  $R$

X02 **REM** OUTPUT  $C$

X10 **LET**  $C = \pi * R * R$

X20 **RETURN**

**3.11 NAME:** Area of Circle, Given the Perimeter  
(Circumference).

**INPUT VARIABLES:**  $P$  = perimeter of circle.

**OUTPUT VARIABLES:**  $C$  = area of circle;  $R$  =  
radius.

X00 **REM** CIRCLE AREA (PERIM)

X01 **REM** INPUT  $P$

X02 **REM** OUTPUT  $C$  AND  $R$

X10 **LET**  $R = P / (2 * \pi)$

X20 **LET**  $C = P * P / (4 * \pi)$

X30 **RETURN**

## AREA OF AN ELLIPSE

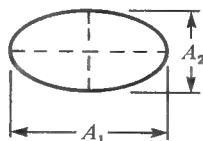
**PURPOSE:** Calculation

**3.12 NAME:** Area of Ellipse.

**INPUT VARIABLES:**  $A1$  = major axis;  $A2$  =  
minor axis.

OUTPUT VARIABLES:  $E$  = area of ellipse.

```
X00 REM ELLIPSE AREA
X01 REM INPUT A1 AND A2
X02 REM OUTPUT E
X10 LET E= PI*A1*A2/4
X20 RETURN
```



## SURFACE AREA OF SOLIDS

PURPOSE: Calculation

**3.13** NAME: Surface Area of Sphere.

INPUT VARIABLES:  $R$  = radius of sphere.

OUTPUT VARIABLES:  $SS$  = surface area of sphere.

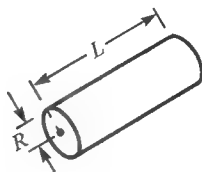
```
X00 REM AREA OF SPHERE
X01 REM INPUT R
X02 REM OUTPUT SS
X10 LET SS=4*PI*R*R
X20 RETURN
```

**3.14** NAME: Surface Area of Right Circular Cylinder (Including Ends).

INPUT VARIABLES:  $R$  = radius of cylinder base;  
 $L$  = length.

OUTPUT VARIABLES:  $SC$  = surface area of cylinder.

```
X00 REM AREA OF CYLINDER
X01 REM INPUT R AND L
X02 REM OUTPUT SC
X10 LET SC=2*PI*R*(R+L)
X20 RETURN
```

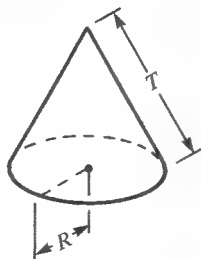


**3.15 NAME:** Surface Area of Right Circular Cone  
(Including Base Area).

**INPUT VARIABLES:**  $R$  = radius of cone base;  $T$   
= distance from vertex to edge of base.

**OUTPUT VARIABLES:**  $AC$  = surface area of  
cone.

```
X00  REM AREA OF CONE
X01  REM INPUT R AND T
X02  REM OUTPUT AC
X10  LET AC=PI *R*(R+T)
X20  RETURN
```



### VOLUME OF A CUBE

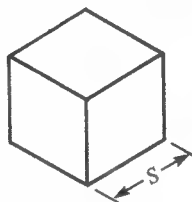
**PURPOSE:** Calculation

**3.16 NAME:** Volume of Cube Given One Side.

**INPUT VARIABLES:**  $S$  = side of cube.

**OUTPUT VARIABLES:**  $CV$  = volume of cube.

```
X00  REM CUBE VOLUME (SIDE)
X01  REM INPUT S
X02  REM OUTPUT CV
X10  LET CV=S**3
X20  RETURN
```

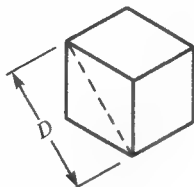


**3.17 NAME:** Volume of Cube Given the Diagonal of a  
Face.

**INPUT VARIABLES:**  $D$  = diagonal.

**OUTPUT VARIABLES:**  $CV$  = volume;  $S$  = side.

```
X00  REM CUBE VOLUME (DIAG)
X01  REM INPUT D
X02  REM OUTPUT CV AND S
X10  LET S=D/SQR2
X20  LET CV=S**3
X30  RETURN
```



## VOLUME OF A SPHERE

**PURPOSE:** Calculation

**3.18 NAME:** Volume of Sphere Given the Radius.

**INPUT VARIABLES:** R = radius.

**OUTPUT VARIABLES:** SV = volume.

X00 **REM** SPHERE VOLUME

X01 **REM** INPUT R

X02 **REM** OUTPUT SV

X10 **LET** SV=(4/3)\*PI\*R\*\*3

X20 **RETURN**

**3.19 NAME:** Volume of Sphere Given the Surface Area.

**INPUT VARIABLES:** SS = surface area of sphere.

**OUTPUT VARIABLES:** SV = volume of sphere.

X00 **REM** SPHERE VOLUME

(AREA)

X01 **REM** INPUT SS

X02 **REM** OUTPUT SV

X10 **LET** SV=SS\*R/3

X20 **RETURN**

## VOLUME OF A CYLINDER AND CONE

**PURPOSE:** Calculation

**3.20 NAME:** Volume of Cylinder.

**INPUT VARIABLES:** R = radius; L = length.

**OUTPUT VARIABLES:** CY = volume of cylinder.

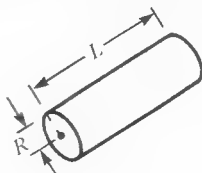
X00 **REM** CYLINDER VOLUME

X01 **REM** INPUT R AND L

X02 **REM** OUTPUT CY

X10 **LET** CY=PI\*R\*R\*L

X20 **RETURN**

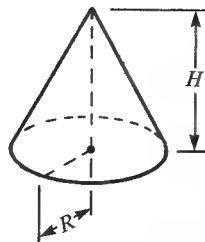


**3.21** NAME: Volume of a Cone.

INPUT VARIABLES: R = radius of base; H = height of cone.

OUTPUT VARIABLES: CN = volume of cone.

```
X00  REM CONE VOLUME
X01  REM INPUT R AND H
X02  REM OUTPUT CN
X10  LET CN=PI*H*R*R/3
X20  RETURN
```





## Chapter 4

# Conversion

Conversion subroutines are usually the most easily understood. A *conversion subroutine* is simply a method for changing the form or base of a value without changing the value itself.

For example, all football fields are 100 yards long. If the yards are converted to feet, the field can be said to be 300 feet long. The *value* of the length of the field has not changed, but the form of the measurement has been converted from yards to feet. If the measurement is converted from feet to inches, then the football field is 3600 inches long. If, as some famous coach has said, football is a game of inches, then it is a game of 3600 inches.

A conversion subroutine is usually a change based upon a relationship or ratio. Consider the relationship between feet and inches. 1 foot equals 12 inches, and 1 inch equals  $1/12$  foot. If we let  $F$  equal the number of feet, and  $I$  equal the number of inches, then:

```
X10  LET F=(1/12)*I
```

```
X20  RETURN
```

and

```
X10  LET I=12*F
```

```
X20  RETURN
```

In the first pair of lines, you supply the number of inches  $I$ , and the subroutine gives you the equivalent number of feet  $F$ .

In the second pair of lines, you supply the number of feet F, and the subroutine gives you the equivalent number of inches I.

You can write your own conversion subroutines for any relationship that can be stated as follows: **LET** X=C\*Y, where C is the conversion constant which changes the form of Y to the form of X.

The following subroutines provide conversions for a large number of common forms of measurement. In each case, the conversions are provided in sets, so that the conversion from one form to another is accompanied by the *reciprocal* conversion, which will convert from the second form back to the original.

In very few cases, such as in Subroutine 4.4 (LIQUID ENGLISH : LITER), a number of conversions are grouped in a single subroutine. Such a multiple conversion is often useful, but be sure to notice that in 4.4a, the **GOSUB** used must match the input variable which you are furnishing. If in the case of Subroutine 4.4a you are furnishing CP (cups) as the input variable, then you must enter the subroutine at line X40, rather than at line X00. In this case, if you do not enter the subroutine at line X40, then the computer will be looking for undefined variables. Likewise, if you are furnishing QT (quarts), then you must enter the subroutine at line X20 (**GOSUB** X20). If the subroutine is not entered correctly, the computer will return an error code 2 (undefined variable used).

Finally, you will notice that the binary : decimal and decimal : hex conversions (4.22, 4.23, and 4.24) are divided into "small-number" conversions and "large-number" conversions. The large-number conversions can be used for any number, but large number conversions use up more memory space and require more processing time.

The Timex/Sinclair machine code works with and recognizes only decimal numbers between 0 and 255. The small-number conversions will handle exactly the same numbers, and are less complex conversions than those required for larger numbers. Thus, the small-number conversions should be used when working with machine code numbers.

## OUNCES : MILLILITERS

**PURPOSE:** Conversion

- 4.1a** NAME: Ounce to Milliliter.  
 INPUT VARIABLES: OZ = ounce.  
 OUTPUT VARIABLES: ML = milliliter.

```
X00 REM OUNCE TO MILLILITER
X01 REM INPUT OZ
X02 REM OUTPUT ML
X10 LET ML=29.586*OZ
X20 RETURN
```

- 4.1b** NAME: Milliliter to Ounce.  
 INPUT VARIABLES: ML = milliliter.  
 OUTPUT VARIABLES: OZ = ounce.

```
X00 REM MILLILITER TO OUNCE
X01 REM INPUT ML
X02 REM OUTPUT OZ
X10 LET OZ=.0338*ML
X20 RETURN
```

## QUARTS : LITERS

**PURPOSE:** Conversion

- 4.2a** NAME: Quart to Liter.  
 INPUT VARIABLES: QT = quart.  
 OUTPUT VARIABLES: LT = liter.

```
X00 REM QUART TO LITER
X01 REM INPUT QT
X02 REM OUTPUT LT
X10 LET LT=.9464*QT
X20 RETURN
```

**4.2b** NAME: Liter to Quart.  
INPUT VARIABLES: LT = liter.  
OUTPUT VARIABLES: QT = quart.

```
X00  REM LITER TO QUART
X01  REM INPUT LT
X02  REM OUTPUT QT
X10  LET QT=1.0567*LT
X20  RETURN
```

## GALLONS : LITERS

**PURPOSE:** Conversion

**4.3a** NAME: Gallon to Liter.  
INPUT VARIABLES: GL = gallon.  
OUTPUT VARIABLES: LT = liter.

```
X00  REM GALLON TO LITER
X01  REM INPUT GL
X02  REM OUTPUT LT
X10  LET LT=3.7854*GL
X20  RETURN
```

**4.3b** NAME: Liter to Gallon.  
INPUT VARIABLES: LT = liter.  
OUTPUT VARIABLES: GL = gallon.

```
X00  REM LITER TO GALLON
X01  REM INPUT LT
X02  REM OUTPUT GL
X10  LET GL=0.2642*LT
X20  RETURN
```

## LIQUID ENGLISH : LITER

**PURPOSE:** Conversion between Liters and a Number of English Measures

**4.4a** **NAME:** Ounces, Cups, Pints, Quarts, and Gallons to Liters.

**INPUT VARIABLES:** OZ = ounce; CP = cup; PT = pint; QT = quart; GL = gallon.

**OUTPUT VARIABLES:** LT = liter (= 1000 millileters).

```

X00  REM LIQUID ENGLISH TO LITER
X01  REM INPUT GOSUB 50=OZ, 40=CP
X02  REM 30=PT, 20=QT, 10=GL
X03  REM OUTPUT LT
X10  LET QT=4*GL
X20  LET PT=2*QT
X30  LET CP=2*PT
X40  LET OZ=8*CP
X50  LET LT=0.02957*OZ
X60  RETURN

```

**NOTE:** This subroutine requires that you enter it *only* at the line appropriate to your input variable. These inputs are: X10 (or X00) for GL, X20 for QT, X30 for PT, X40 for CP, and X50 for OZ.

**4.4b** **NAME:** Liters to Ounces, Cups, Pints, Quarts, and Gallons.

**INPUT VARIABLES:** LT = liters (= 1000 millileters).

**OUTPUT VARIABLES:** OZ = ounce; CP = cup; PT = pint; QT = quart; GL = gallon.

```

X00  REM LITER TO LIQUID ENGLISH
X01  REM INPUT LT

```

```

X02  REM OUTPUT OZ, CP, PT, QT, AND GL
X10  LET GL=0.2642*LT
X20  LET QT=4*GL
X30  LET PT=2*QT
X40  LET CP=2*PT
X50  LET OZ=8*CP
X60  RETURN

```

## INCHES : CENTIMETERS

**PURPOSE:** Conversion

**4.5a** NAME: Inches to Centimeters.  
INPUT VARIABLES: IN = inches.  
OUTPUT VARIABLES: CM = centimeters.

```

X00  REM INCHES TO CENTIMETERS
X01  REM INPUT IN
X02  REM OUTPUT CM
X10  LET CM=2.54*IN
X20  RETURN

```

**4.5b** NAME: Centimeters to Inches  
INPUT VARIABLES: CM = centimeters.  
OUTPUT VARIABLES: IN = inches.

```

X00  REM CENTIMETERS TO INCHES
X01  REM INPUT CM
X02  REM OUTPUT IN
X10  LET IN=0.3937*CM
X20  RETURN

```

**FEET : METERS****PURPOSE:** Conversion

- 4.6a** NAME: Feet to Meters.  
 INPUT VARIABLES: FT = feet.  
 OUTPUT VARIABLES: MT = meters.

X00 **REM FEET TO METERS**X01 **REM INPUT FT**X02 **REM OUTPUT MT**X10 **LET MT=0.3048\*FT**X20 **RETURN**

- 4.6b** NAME: Meters to Feet.  
 INPUT VARIABLES: MT = meters.  
 OUTPUT VARIABLES: FT = feet.

X00 **REM METERS TO FEET**X01 **REM INPUT MT**X02 **REM OUTPUT FT**X10 **LET FT=3.2808\*MT**X20 **RETURN****YARDS : METERS****PURPOSE:** Conversion

- 4.7a** NAME: Yards to Meters.  
 INPUT VARIABLES: YD = yards.  
 OUTPUT VARIABLES: MT = meters.

X00 **REM YARDS TO METERS**X01 **REM INPUT YD**X02 **REM OUTPUT MT**X10 **LET MT=0.9144\*YD**X20 **RETURN**

**4.7b** NAME: Meters to Yards.

INPUT VARIABLES: MT = meters.

OUTPUT VARIABLES: YD = yards.

X00 REM METERS TO YARDS

X01 REM INPUT MT

X02 REM OUTPUT YD

X10 LET YD=1.0936\*MT

X20 RETURN

## INCHES, FEET, YARDS : METERS

**PURPOSE:** Conversion between Meters and a  
Number of English Measurements

**4.8a** NAME: Inches, Feet, and Yards to Meters.

INPUT VARIABLES: IN = inches; FT = feet;  
YD = yards.

OUTPUT VARIABLES: MT = meters.

X00 REM INCH,FOOT,YARD TO METER

X01 REM INPUT GOSUB 30=IN,

X02 REM 20=FT, AND 10=YD

X03 REM OUTPUT MT

X10 LET FT=3\*YD

X20 LET IN=12\*FT

X30 LET MT=0.0254\*IN

X40 RETURN

**NOTE:** This subroutine requires that you enter it *only* at the line appropriate to your input variable. These inputs are: X10 (or X00) for YD, X20 for FT, and X30 for IN.



- 4.8b** NAME: Meters to Inches, Feet, and Yards.  
 INPUT VARIABLES: MT = meters.  
 OUTPUT VARIABLES: IN = inches; FT = feet;  
 YD = yards.

```

X00  REM METER TO INCH, FOOT, YARD
X01  REM INPUT MT
X02  REM OUTPUT IN, FT, YD
X10  LET YD=1.0936*MT
X20  LET FT=3*YD
X30  LET IN=12*FT
X40  RETURN
  
```

## MILES : KILOMETERS

**PURPOSE:** Conversion

- 4.9a** NAME: Statute Miles to Kilometers.  
 INPUT VARIABLES: MI = miles.  
 OUTPUT VARIABLES: KM = kilometers.

```

X00  REM MILES TO KILOMETERS
X01  REM INPUT MI
X02  REM OUTPUT KM
X10  LET KM=1.6093*MI
X20  RETURN
  
```

- 4.9b** NAME: Kilometers to Statute Miles.  
 INPUT VARIABLES: KM = kilometers.  
 OUTPUT VARIABLES: MI = miles.

```

X00  REM KILOMETERS TO MILES
X01  REM INPUT KM
X02  REM OUTPUT MI
X10  LET MI=0.621*KM
X20  RETURN
  
```

## FATHOMS : FEET (and Meters)

**PURPOSE:** Conversion

**4.10a** NAME: Fathom to Feet and Meters.

INPUT VARIABLES: FM = fathom.

OUTPUT VARIABLES: FT = feet; MT = meters.

X00 **REM FATHOM TO FEET AND METER**

X01 **REM INPUT FM**

X02 **REM OUTPUT FT AND MT**

X10 **LET FT=6\*FM**

X20 **LET MT=1.829\*FM**

X30 **RETURN**

**4.10b** NAME: Feet to Fathom.

INPUT VARIABLES: FT = feet.

OUTPUT VARIABLES: FM = fathom.

X00 **REM FEET TO FATHOM**

X01 **REM INPUT FT**

X02 **REM OUTPUT FM**

X10 **LET FM=FT/6**

X20 **RETURN**

## FURLONGS : MILES (and Kilometers)

**PURPOSE:** Conversion

**4.11a** NAME: Furlong to Statute Miles and Kilometers.

INPUT VARIABLES: FL = furlong.

OUTPUT VARIABLES: MI = miles; KM =  
kilometers.

X00 **REM FURLONG TO MI AND KM**

X01 **REM INPUT FL**

```

X02  REM OUTPUT MI AND KM
X10  LET MI=FL/8
X20  LET KM=1.6093*MI
X30  RETURN

```

**4.11b** NAME: Statute Miles to Furlongs.  
 INPUT VARIABLES: MI = miles.  
 OUTPUT VARIABLES: FL = furlongs.

```

X00  REM MILES TO FURLONGS
X01  REM INPUT MI
X02  REM OUTPUT FL
X10  LET FL=8*MI
X20  RETURN

```

## NAUTICAL MILES : STATUTE MILES AND KILOMETERS

**PURPOSE:** Conversion

**4.12a** NAME: Nautical Mile to Statute Mile and  
Kilometers.

INPUT VARIABLES: NM = nautical mile.

OUTPUT VARIABLES: MI = statute mile; KM =  
kilometer.

```

X00  REM NAUTICAL MILES TO
X01  REM MILES AND KILOMETERS
X02  REM INPUT NM
X03  REM OUTPUT MI AND KM
X10  LET MI=1.150779*NM
X20  LET KM=1.852*NM
X30  RETURN

```

**4.12b** NAME: Kilometers and Statute Miles to Nautical Miles.

INPUT VARIABLES: KM = kilometer; MI = statute miles.

OUTPUT VARIABLES: NM = nautical miles.

```

X00  REM MILES AND KILOMETERS
X01  REM TO NAUTICAL MILES
X02  REM INPUT GOSUB X10=MI, X20=KM
X03  REM OUTPUT NM
X10  LET KM=1.6093*MI
X20  LET NM=0.54*KM
X30  RETURN

```

NOTE: This subroutine must be entered only at the line number appropriate for your input variable. Input lines are X10 (or X00) for MI, and X20 for KM.

**LIGHT YEARS : KILOMETERS AND MILES**

**PURPOSE:** Conversion

**4.13a** NAME: Light Years to Kilometers and Miles.

INPUT VARIABLES: LY = light year.

OUTPUT VARIABLES: KM = kilometer; MI = miles.

```

X00  REM LIGHT YEARS TO
X01  REM KILOMET AND MILES
X02  REM INPUT LY
X03  REM OUTPUT KM AND MI
X10  LET MI=5.87851E12*LY
X20  LET KM=1.609345*MI
X30  RETURN

```

**4.13b** NAME: Miles and Kilometers to Light Years.

INPUT VARIABLES: MI = miles; KM = kilometers.

OUTPUT VARIABLES: LY = light year.

X00 REM KILOMET AND MILES

X01 REM TO LIGHT YEARS

X02 REM INPUT GOSUB 10=KM,20=MI

X03 REM OUTPUT LY

X10 LET MI=0.621371\*KM

X20 LET LY=1.701111E-13\*MI

X30 RETURN

## FEET PER SECOND : MILES PER HOUR

PURPOSE: Conversion

**4.14a** NAME: Feet per Second to Miles per Hour.

INPUT VARIABLES: FS = feet per second.

OUTPUT VARIABLES: MH = miles per hour.

X00 REM FEET/SEC TO MILES/HOUR

X01 REM INPUT FS

X02 REM OUTPUT MH

X10 LET MH=0.6818\*FS

X20 RETURN

**4.14b** NAME: Miles per Hour to Feet per Second.

INPUT VARIABLES: MH = miles per hour.

OUTPUT VARIABLES: FS = feet per second.

X00 REM MILES/HOUR TO FEET/SEC

X01 REM INPUT MH

X02 REM OUTPUT FS

X10 LET FS=1.4667\*MH

X20 RETURN

## METERS PER SECOND : KILOMETERS PER HOUR

**PURPOSE:** Conversion

- 4.15a** NAME: Meters per Second to Kilometers per Hour.  
INPUT VARIABLES: MS = meters per second.  
OUTPUT VARIABLES: KH = kilometers per hour.

```
X00  REM METERS/SEC TO KILOMET/HOUR
X01  REM INPUT MS
X02  REM OUTPUT KH
X10  LET KH=3.6*MS
X20  RETURN
```

- 4.15b** NAME: Kilometers per Hour to Meters per Second.  
INPUT VARIABLES: KH = kilometers per hour.  
OUTPUT VARIABLES: MS = meters per second.

```
X00  REM KILOMET/HR TO METER/SEC
X01  REM INPUT KH
X02  REM OUTPUT MS
X10  LET MS=0.2778*KH
X20  RETURN
```

## MILES PER HOUR : KILOMETERS PER HOUR

**PURPOSE:** Conversion

- 4.16a** NAME: Miles per Hour to Kilometers per Hour.  
INPUT VARIABLES: MH = miles per hour.  
OUTPUT VARIABLES: KH = kilometers per hour.

```
X00  REM MILES/HR TO KMETERS/HR
X01  REM INPUT MH
X02  REM OUTPUT KH
```

X10 **LET** KH=1.6093\*MH

X20 **RETURN**

- 4.16b** NAME: Kilometers per Hour to Miles per Hour.  
 INPUT VARIABLES: KH = kilometers per hour.  
 OUTPUT VARIABLES: MH = miles per hour.

X00 **REM** KMETERS/HR TO MILES/HR

X01 **REM** INPUT KH

X02 **REM** OUTPUT MH

X10 **LET** MH=0.6214\*KH

X20 **RETURN**

## ACRES : SQUARE KILOMETERS

**PURPOSE:** Conversion

- 4.17a** NAME: Acres to Square Kilometers.  
 INPUT VARIABLES: AC = acre.  
 OUTPUT VARIABLES: SK = square kilometers.

X00 **REM** ACRE TO SQ KILOMETER

X01 **REM** INPUT AC

X02 **REM** OUTPUT SK

X10 **LET** SK=0.004047\*AC

X20 **RETURN**

- 4.17b** NAME: Square Kilometers to Acres.  
 INPUT VARIABLES: SK = square kilometer.  
 OUTPUT VARIABLES: AC = acre.

X00 **REM** SQ KILOMETER TO ACRE

X01 **REM** INPUT SK

X02 **REM** OUTPUT AC

X10 **LET** AC=247.1\*SK

X20 **RETURN**

## SQUARE YARDS : SQUARE METERS

**PURPOSE:** Conversion

- 4.18a** NAME: Square Yards to Square Meters.  
INPUT VARIABLES: SY = square yards.  
OUTPUT VARIABLES: SM = square meters.

```
X00  REM SQ YARDS TO SQ METERS
X01  REM INPUT SY
X02  REM OUTPUT SM
X10  LET SM=0.8361*SY
X20  RETURN
```

- 4.18b** NAME: Square Meters to Square Yards.  
INPUT VARIABLES: SM = square meters.  
OUTPUT VARIABLES: SY = square yards.

```
X00  REM SQ METERS TO SQ YARDS
X01  REM INPUT SM
X02  REM OUTPUT SY
X10  LET SY=1.196*SM
X20  RETURN
```

## SQUARE YARDS : ACRES

**PURPOSE:** Conversion

- 4.19a** NAME: Square Yard to Acre.  
INPUT VARIABLES: SY = square yard.  
OUTPUT VARIABLES: AC = acre.

```
X00  REM SQ YARD TO ACRE
X01  REM INPUT SY
X02  REM OUTPUT AC
X10  LET AC=2.066E-4*SY
X20  RETURN
```



**4.19b** NAME: Acre to Square Yard.

INPUT VARIABLES: AC = acre.

OUTPUT VARIABLES: SY = square yard.

X00 REM ACRE TO SQ YARD

X01 REM INPUT AC

X02 REM OUTPUT SY

X10 LET SY = 4840\*AC

X20 RETURN

## CELSIUS : FAHRENHEIT

**PURPOSE:** Conversion

**4.20a** NAME: Celsius to Fahrenheit.

INPUT VARIABLES: C = celsius temperature.

OUTPUT VARIABLES: F = fahrenheit temperature.

X00 REM CELSIUS TO FAHRENHEIT

X01 REM INPUT C

X02 REM OUTPUT F

X10 LET F=1.8\*C+32

X20 RETURN

**4.20b** NAME: Fahrenheit to Celsius.

INPUT VARIABLES: F = fahrenheit temperature.

OUTPUT VARIABLES: C = celsius temperature.

X00 REM FAHRENHEIT TO CELSIUS

X01 REM INPUT F

X02 REM OUTPUT C

X10 LET C=5\*(F-32)/9

X20 RETURN

**DEGREES : RADIANS : GRADS****PURPOSE:** Conversion**4.21a** NAME: Degrees to Radians and Grads.

INPUT VARIABLES: D = degree.

OUTPUT VARIABLES: R = radian; G = grad.

X00 **REM** DEGREE **TO** Radian-GRADX01 **REM** INPUT DX02 **REM** OUTPUT R **AND** GX10 **LET** R=PI\*D/180X20 **LET** G=10\*D/9X30 **RETURN****4.21b** NAME: Grads to Radians and Degrees.

INPUT VARIABLES: G = grads.

OUTPUT VARIABLES: R = radians;

D = degrees.

X00 **REM** GRAD **TO** Radian-DEGREEX01 **REM** INPUT GX02 **REM** OUTPUT R **AND** DX10 **LET** D=9\*G/10X20 **LET** R =PI\*G/200X30 **RETURN****4.21c** NAME: Radians to Degrees and Grads.

INPUT VARIABLES: R = radians.

OUTPUT VARIABLES: D = degrees; G = grads.

X00 **REM** Radian **TO** DEGREE-GRADX01 **REM** INPUT RX02 **REM** OUTPUT D **AND** GX10 **LET** D=180\*R/PIX20 **LET** G=200\*R/PIX30 **RETURN**

**BINARY : DECIMAL (SMALL NUMBERS)**

**PURPOSE:** To Convert from One Number Base to Another

**4.22a NAME:** Binary to Decimal (Base 2 to Base 10).

**INPUT VARIABLES:** B\$ = binary number; 8 digits long.

**OUTPUT VARIABLES:** D = decimal equivalent of B\$.

```
X00  REM BINARY TO DECIMAL
X01  REM INPUT B$
X02  REM OUTPUT D
X10  LET D=0
X20  FOR I=1 TO 8
X30  IF B$(I)="1" THEN LET D=D+(2**(8-I))
X40  NEXT I
X50  RETURN
```

**4.22b NAME:** Decimal to Binary (Base 10 to Base 2).

**INPUT VARIABLES:** D = decimal Number <= 255.

**OUTPUT VARIABLES:** B\$ = binary equivalent of D.

```
X00  REM DECIMAL TO BINARY
X01  REM INPUT D
X02  REM OUTPUT B$
X10  LET B$=""
X20  FOR I=1 TO 8
X30  LET B$=B$+STR$ INT(D/(2**(8-I)))
X40  LET D=D-(2**(8-I))*INT(D/(2**(8-I)))
X50  NEXT I
X60  RETURN
```

## DECIMAL : HEX (SMALL NUMBERS)

**PURPOSE:** To Convert from One Number Base to Another.

**4.23a NAME:** Decimal to Hex (Base 10 to Base 16).

**INPUT VARIABLES:** D = decimal number not greater than 225.

**OUTPUT VARIABLES:** H\$ = hexadecimal equivalent of D.

```
X00  REM DECIMAL TO HEX
X01  REM INPUT D
X02  REM OUTPUT H$
X10  LET H$=""
X20  LET H$=CHR$(28+INT(D/16))
X30  LET H$=H$+CHR$(28+D-16*INT (D/16))
X40  RETURN
```

**4.23b NAME:** Hex to Decimal (Base 16 to Base 10).

**INPUT VARIABLES:** H\$ = a two-character hexadecimal number.

**OUTPUT VARIABLES:** D = the decimal equivalent of H\$.

```
X00  REM HEX TO DECIMAL
X01  REM INPUT H$
X02  REM OUTPUT D
X10  LET D=16*(CODE H$-28)+CODE
      H$(2 TO 2)-28
X20  RETURN
```

**DECIMAL : HEX (LARGE NUMBERS)**

**PURPOSE:** To Convert a Number between Base 10  
and Base 16

**4.24a NAME:** Decimal to Hexadecimal.

**INPUT VARIABLES:** D = decimal (base 10)  
number.

**OUTPUT VARIABLES:** H\$ = hex (base 16)  
number.

```

X00  REM DEC TO HEX(LARGE)
X01  REM INPUT D
X02  REM OUTPUT H$
X10  LET N= LN D/LN 16
X15  DIM H$(1+INT N)
X20  FOR I=1 TO 1+INT N
X25  LET H$(I)="0"
X30  NEXT I
X40  LET H=LN D/LN 16
X50  LET H$(1+INT N-INT
      H)=CHR$(28+INT(D/16**INT H))
X60  LET D=D-(16**INT H)*INT(D/16**INT H)
X70  IF D=0 THEN RETURN
X80  GOTO X40

```

**4.24b NAME:** Hexadecimal To Decimal.

**INPUT VARIABLES:** H\$ = hex (base 16) number.

**OUTPUT VARIABLES:** D = decimal (base 10)  
number.

```

X00  REM HEX TO DEC(LARGE)
X01  REM INPUT H$
X02  REM OUTPUT D
X10  LET D=0

```

```

X20  LET N=LEN H$
X30  FOR I=N TO 1 STEP -1
X40  LET D=D+(CODE H$(I TO
      I)-28)*(16**(N-I))
X50  NEXT I
X60  RETURN

```

## DECIMAL BASE TO ANY BASE

**PURPOSE:** To Convert a Number in Base 10  
(Decimal Number) to a Number in any other Base

**4.25 NAME:** Decimal to other Base.

**INPUT VARIABLES:** D = decimal number to be  
converted; B = base to which D is to be  
converted.

**OUTPUT VARIABLES:** A\$ = number in base B  
equivalent to D in base 10.

```

X00  REM BASE CONVERSION
X01  REM INPUT D AND B
X02  REM OUTPUT A$
X10  LET N=LN D/LN B
X15  DIM A$(1+INT N)
X20  FOR I=1 TO 1+INT N
X30  LET A$(I)="0"
X40  NEXT I
X50  LET H=LN D/LN B
X60  LET A$(1+INT N-INT
      H)=CHR$(28+INT(D/B**INT H))
X70  LET D=D-(B**INT H)*INT(D/B**INT H)
X80  IF D=0 THEN RETURN
X90  GOTO X50

```

# Chapter 5

## Statistics

The statistical procedures in this chapter are used in many disciplines, but are especially useful in the behavioral sciences. Although the procedures described in this chapter are relatively simple, they can be very useful and can save much manual mathematics once the initial requirements are met.

Almost every one of these subroutines can be modified to be an independent program. However, they may present the best possible use when linked together into a complete, flexible program which will meet your needs. For example, information stored in strings (as shown in Chapter 6) or information stored as an array can be arranged in sequence by using the subroutines found in Chapter 7. This sequenced information can then be processed by using the subroutines in this chapter. The information can then be plotted using subroutines from Chapter 9, or can be listed in a table as shown in Chapter 10. Your task is to determine what you require, and how to link the subroutines together in order to obtain the results you want.

The first two subroutines in this chapter will accept data and condense that data into a frequency distribution. As with many other subroutines in this book, the data must first be converted into the form of an array. What follows is a program that will accept your input and place it into a properly dimensioned array:

```

1  REM "ARRAY INPUT"
10 LET MX=0
20 PRINT "INPUT_NUMBER_OF_ITEMS_IN_
   DATA"
25 INPUT N
30 DIM A(N)
35 CLS
40 FOR I=1 TO N
50 PRINT "A(";I;")_=_";
55 INPUT A(I)
60 IF I=1 THEN LET MN=A(I)
65 PRINT A(I)
70 IF A(I)>MX THEN LET MX=A(I)
75 IF A(I)<MN THEN LET MN=A(I)
80 GOSUB Y000 [To AUTO-SCROLL, Subroutine 8.6]
90 NEXT I

```

This program places your data into an array, A(I), and also provides N (the number of items in the array), MX (the maximum value in the array), and MN (the minimum value in the array). If MX and MN are not needed, you may wish to delete lines 10, 60, 70, and 75.

The purpose of line 80 is to avoid overfilling the screen when you have more than 22 items to input. Although line 80 is not necessary, it (and the **AUTO-SCROLL** subroutine to which it goes) will help avoid getting a 5 error code, and will allow you to concentrate on inputting your data.

As we have said, the first two subroutines in this chapter use data in the form of an array and will provide a frequency distribution of the data. A frequency distribution is a very useful way to group data. An example may make this usefulness more obvious. A frequency distribution lists a value (or class, or score) and the number of times (or the *frequency* with which) the value occurs in your data.



In the output of both of the frequency distribution subroutines in this chapter, the  $B(I,1)$  array is a list of the values and the  $B(I,2)$  array is a list of the corresponding frequencies of the values. For example, suppose that your data happened to be: 0, 4, 9, 5, 0, 5, 8, 6, 5, 5, 7, 2, 8, 6, 6. To be used in the subroutines the data must be in the form of an array such as the one shown in Table 5-1, where there are 15 values ( $N=15$ ) with the maximum value equal to 9 ( $MX=9$ ) and the minimum value equal to 0 ( $MN=0$ ). In this data, there are 2 zeros, so it can be said that the frequency of zero is 2. Likewise, the frequency of ones is 0 (there are no ones), the frequency of twos is 1 (there is 1 two), and so on. If we enter this data into subroutine 5.1, the outcome could be printed to show how many of each score (the frequency) there are in the data. The printout would look like Table 5-2.

Table 5-2 shows that in the data there are 2 zeros, 0 ones, 1 two, 0 threes, 1 four, 4 fives, and so on. This can be seen by looking first at the *frequency* column  $B(I,2)$ , and then referring to the score column  $B(I,1)$ . In other words, the first column of Table 5-2 is a listing of the values, and the second column is a listing of the frequency of these values in the data.

The data array shown in Table 5-1 could be used with Subroutine 5.2 with slightly different results. In order to use Subroutine 5.2, however, we must first decide upon an "interval width," which is the number of scores to be grouped. An interval width of 5 would group the scores 1-5, 6-10, 11-15, etc. An interval width of 2 would group 1-2, 3-4, 5-6, etc. If we choose an interval width of 1 and run Subroutine 5.2, we would expect to get data as shown in Table 5-3.

**Table 5-1**

$A(1)=0$	$A(6)=5$	$A(11)=7$
$A(2)=4$	$A(7)=8$	$A(12)=2$
$A(3)=9$	$A(8)=6$	$A(13)=8$
$A(4)=5$	$A(9)=5$	$A(14)=6$
$A(5)=0$	$A(10)=5$	$A(15)=6$

Table 5-2

<b>B(I,1)</b> <b>(score)</b>	<b>B(I,2)</b> <b>(frequency)</b>
0	2
1	0
2	1
3	0
4	1
5	4
6	3
7	1
8	2
9	1

Table 5-3

<b>B(I,1)</b> <b>(upper limit)</b>	<b>B(I,2)</b> <b>(frequency)</b>
9.5	1
8.5	2
7.5	1
6.5	3
5.5	4
4.5	1
3.5	0
2.5	1
1.5	0
0.5	2

Because the interval is 1 in the example shown in Table 5-3, the information is the same as the information obtained from Subroutine 5.1, except that in this case the values in

$B(I,1)$  are given as “upper real limits.” The “upper real limit” means that instead of simply measuring the frequency at a single value, such as 6, we measure the frequency over a range of values, such as from 5.5 to 6.5. The interval of 1 ( $6.5 - 5.5 = 1$ ) determines the range of the value over which we will measure the frequency.

If we choose an interval width of 3, the result will be as shown in Table 5-4, where the frequency of scores between 9.5 and 6.5 is 4. This listing represents the same data as used in Tables 5-1, 5-2, and 5-3, but here the data is grouped in larger clumps. For further information about frequency distributions, you may want to obtain an elementary statistics book from your library.

Subroutine 5.3 calculates the mean, median, and mode for an array of scores. As noted, however, the median may be slightly in error if there is more than one score equal to the median. Subroutine 5.4 provides an accurate median value for grouped frequency distributions.

The two “Z Score” subroutines in this chapter (5.8 and 5.9) calculate Z scores. The difference between them is that while Subroutine 5.9 is a complete subroutine within itself, Subroutine 5.8 can be used only in a program which also contains the Variance and Standard Deviation Subroutine (5-7). If you have no other need to calculate the variance or standard deviation in your program (and therefore no other need for Subroutine 5-7), then use only the Subroutine 5-9 to obtain Z scores.

**Table 5-4**

<b>B(I,1) (upper limit)</b>	<b>B(I,2) (frequency)</b>
9.5	4
6.5	8
3.5	1
0.5	2

The last subroutine in this chapter (5.10), titled **LINEAR REGRESSION**, can be extremely useful. Using paired values such as height-to-weight or advertising dollars to sales income (or any  $x$ -coordinate-to- $y$ -coordinate "bivariant" value), this subroutine will calculate the parameters of a prediction line. The pairs of input data for Subroutine 5.10 must be in the form of a two-dimensional array.

What follows is a program which will allow you to input a set of paired values to obtain the required array. The two halves of the set are identified as **X** and **Y**:

```

1  REM "X-Y INPUT"
10 PRINT "INPUT NUMBER OF PAIRS IN
    DATA"
20 INPUT N
30 DIM B(N,2)
35 CLS
40 FOR I=1 TO N
50 PRINT "X(";I;")_ = _";
55 INPUT B(I,1)
60 PRINT B(I,1),"Y(";I;")_ = _";
65 INPUT B(I,2)
70 PRINT B(I,2)
80 GOSUB Y000 [To AUTO-SCROLL, Subroutine 8.6]
90 NEXT I

```

Again, as in the **ARRAY INPUT** program given earlier in this chapter, line 80 goes to the **AUTO-SCROLL** subroutine (8.6) and will keep the screen from filling and giving a 5 error code.

The **X-Y INPUT** program above will provide the input array required for Subroutine 5.10. If the paired values of this array have a linear relationship (if plotted, they will fall in a relatively straight line), the calculated prediction line will accurately estimate where other values can be predicted to fall. The value **R2** (the coefficient of determination) provides an indication of the accuracy of the prediction.

## DISTRIBUTION OF SCORES

**PURPOSE:** To Condense a Set of Raw Scores into a Frequency Distribution

**5.1 NAME:** Frequency Distribution of Scores.

**INPUT VARIABLES:** A(I) = array of scores; N = number of scores in the array; MX = maximum score; MN = minimum score.

**OUTPUT VARIABLES:** B(I,1) = array of score values; B(I,2) = frequency of score B(I,1) in array A(I); R = number of score values.

```

X00  REM FREQUENCY DISTRIBUTION
X01  REM INPUT A(I), N, MX, AND MN
X02  REM OUTPUT B(I,1),B(I,2), AND R
X10  LET R=MX-MN+1
X20  DIM B(R,2)
X30  FOR I=MN TO MX
X40  LET B(I-MN+1,1)=I
X50  FOR J=1 TO N
X60  IF A(J)=B(I-MN+1,1) THEN LET
      B(I-MN+1,2)=B(I-MN+1,2)+1
X70  NEXT J
X80  NEXT I
X90  RETURN

```

**NOTE 1:** MX and MN can be obtained by using the MAX/MIN/MEAN subroutine from Chapter 7, or MX and MN may be set arbitrarily if MX is set equal to or greater than the maximum score and MN is set equal to or less than the minimum score.

**NOTE 2:** All scores are assumed to be integer values.

**PURPOSE:** To Condense a Set of Raw Scores into a Grouped Frequency Distribution

- 5.2 NAME:** Frequency Distribution of Grouped Scores.  
**INPUT VARIABLES:** A(I) = array of scores; N = number of scores in A(I); MX = maximum score; MN = minimum score; W = interval width of grouping.  
**OUTPUT VARIABLES:** B(I,1) = array of upper real limits of interval values; B(I,2) = array of frequency of scores within group; R = number of groups in B(I,1).

```

X00  REM GROUPED FREQ DISTR
X01  REM INPUT A(I),N,MX,MN,W
X02  REM OUTPUT B(I,1),B(I,2),R
X10  LET R=INT((MX-MN)/W)+1
X15  DIM B(R,2)
X20  LET T=MX+.5
X25  LET B=T-W
X30  FOR I=1 TO R
X35  LET B(I,1)=T
X40  FOR J=1 TO N
X50  IF A(J)<B OR A(J)>T THEN GOTO X60
X55  LET B(I,2)=B(I,2)+1
X60  NEXT J
X70  LET B=B-W
X75  LET T=T-W
X80  NEXT I
X90  RETURN

```

**NOTE:** See Note 1 in Subroutine 5.1.

## MEAN, MEDIAN, MODE

**PURPOSE:** To Provide Central Tendency  
Characteristics of a Number Array

- 5.3 NAME:** Mean, Median, and Mode.  
**INPUT VARIABLES:** A(J) = ordered list of  
 scores; N = number of scores in list.  
**OUTPUT VARIABLES:** M1 = mean; M2 =  
 median; M3 = mode; K = number of modes.

```

X00  REM MEAN, MEDIAN, MODE
X01  REM INPUT A(J) AND N
X02  REM OUTPUT M1, M2, M3, AND K
X10  LET SUM=0
X15  LET M3=0
X20  LET K=0
X25  FOR I=1 TO N
X30  LET SUM=SUM+A(I)
X35  IF A(I)=M3 THEN LET K=K+1
X40  IF A(I)>M3 THEN LET K=1
X50  IF A(I)>M3 THEN LET M3=A(I)
X60  NEXT I
X65  LET M1=SUM/N
X70  IF N/2<>INT (N/2) THEN LET M2=A(INT
    (N/2)+1)
X80  IF N/2=INT (N/2) THEN LET M2=(A(N/
    2)+A(1+N/2))/2
X90  RETURN
  
```

**NOTE:** If more than one score in A(J) is equal to the median (M2), the value of M2 may be slightly in error. For an equal or better median value in all cases, see Subroutine 5.4.

**5.4 NAME:** Median for Grouped Frequency Distributions.

**INPUT VARIABLES:**  $B(I,1)$  = array of upper real limits of possible scores or of possible score groups;  $B(I,2)$  = array of frequency of scores in  $B(I,1)$ ;  $R$  = number of scores or groups in  $B(I,1)$ .

**OUTPUT VARIABLES:**  $M2$  = Median Value.

```

X00  REM MEDIAN
X01  REM INPUT B(I,1), B(I,2),
X02  REM AND R
X03  REM OUTPUT M2
X10  LET N=0
X20  LET CF=0
X30  LET W=B(2,1)-B(1,1)
X40  FOR I=1 TO R
X50  IF I<=R/2 THEN LET CF=CF+B(I,2)
X60  LET N=N+B(I,2)
X70  NEXT I
X80  LET M2=B(R/2,1)+W*(0.5*N-CF)/B(R/
    2+1,2)
X90  RETURN

```

**NOTE:**  $B(I,1)$ ,  $B(I,2)$ , and  $R$  may be obtained by using the **GROUPED FREQUENCY DISTRIBUTION** subroutine (5.2).

## PERCENTILE RANK

**PURPOSE:** Calculation of Percentile Rank of a Single Score

**5.5 NAME:** Percentile Rank in a Nongrouped Frequency Distribution.



**INPUT VARIABLES:**  $A(I)$  = array of scores;  $N$  = number of scores in the array;  $X$  = score to be ranked.

**OUTPUT VARIABLES:**  $PR$  = percentile rank of score  $X$ ;  $CF$  = cumulative frequency of position of score  $X$ .

```

X00  REM PERCENTILE RANK
X01  REM INPUT A(I),N, AND X
X02  REM OUTPUT PR AND CF
X10  LET R=0
X20  LET CF=0
X30  FOR I=1 TO N
X40  IF A(I)<X THEN LET CF=CF+1
X50  IF A(I)=X THEN LET R=R+1
X60  NEXT I
X70  LET CF=CF+R/2
X80  LET PR=100*CF/N
X90  RETURN

```

### 5.6 NAME: Percentile Rank in a Grouped Frequency Distribution of Scores.

**INPUT VARIABLES:**  $B(I,A)$  = array of score groups and group upper limit values;  $R$  = number of groups;  $X$  = score to be ranked.

**OUTPUT VARIABLES:**  $PR$  = percentile rank of  $X$ ;  $CF$  = cumulative frequency to score  $X$ .

```

X00  REM GROUP PERCENTILE
X01  REM INPUT B(I,A),R,AND X
X02  REM OUTPUT PR AND CF
X10  LET W=B(2,1)-B(1,1)
X15  LET CF=0
X20  LET N=0

```

```

X30  FOR I=1 TO R
X35  LET N=N+B(I,2)
X40  IF B(I,1)<X THEN LET CF=CF+B(I,2)
X50  IF B(I,1)>=X AND B(I,1)-W<X THEN
      GOTO X80
X60  NEXT I
X65  LET PR=100*CF/N
X70  RETURN
X80  LET CF=CF+((X-B(I-1)+W)/W)*B(I,2)
X90  GOTO X60

```

NOTE: Input variable array B(I,1) is the upper limit value of group I and B(I,2) is the frequency of scores in group B(I,1). See Subroutine 5.2 for score grouping.

## VARIANCE AND STANDARD DEVIATION

**PURPOSE:** To Calculate the Variance and Standard Deviation for a Set of Values

**5.7 NAME:** Variance and Standard Deviation.

**INPUT VARIABLES:** A(I) = array of values; N = number of values in the array.

**OUTPUT VARIABLES:** AVG = mean of values;  
VAR = variance of values; STD = standard deviation.

```

X00  REM VAR AND STD DEVIATION
X01  REM INPUT A(I) AND N
X02  REM OUTPUT AVG, VAR, AND STD
X10  LET SUM=0
X15  LET SQS=0
X20  FOR I=1 TO N
X30  LET SUM=SUM+A(I)

```

```

X40  LET SQS=SQS+A(I)*A(I)
X50  NEXT I
X60  LET AVG=SUM/N
X70  LET VAR=(SQS/N)-AVG*AVG
X80  LET STD=SQR VAR
X90  RETURN

```

## Z SCORE

**PURPOSE:** To Calculate the Z Score Equivalent of an Array of Values

### 5.8 NAME: Z Score (Using GOSUB).

**INPUT VARIABLES:** A(I) = array of values; N = number of values in array; Y00 = line number of subroutine for variance and standard deviation (see Subroutine 5.7).

**OUTPUT VARIABLES:** Z(I) = array of Z scores.

```

X00  REM Z SCORE (GOSUB)
X01  REM INPUT A(I) AND N
X02  REM OUTPUT Z(I)
X10  DIM Z(N)
X20  GOSUB Y00      [to Subroutine 5.7]
X30  FOR I=1 TO N
X40  LET Z(I)=(A(I)-AVG)/STD
X50  NEXT I
X60  RETURN

```

### 5.9 NAME: Z Score (without Using GOSUB).

**INPUT VARIABLES:** A(I) = array of values; N = number of values in the array.

**OUTPUT VARIABLES:** Z(I) = array of Z scores.

```

X00  REM Z SCORE

```

```

X01  REM INPUT A(I) AND N
X02  REM OUTPUT Z(I)
X10  DIM Z(N)
X15  LET SUM=0
X20  LET SQS=0
X25  FOR I=1 TO N
X30  LET SUM=SUM+A(I)
X35  LET SQS=SQS+A(I)*A(I)
X40  NEXT I
X45  LET AVG=SUM/N
X50  LET STD=SQR((SQS/N)-AVG*AVG)
X60  FOR I=1 TO N
X70  LET Z(I)=(A(I)-AVG)/STD
X80  NEXT I
X90  RETURN

```

NOTE: The expression **SQR** on line X50 is the square root function, obtained by pressing the H key while the computer is in the **F** mode.

## LINEAR REGRESSION

**PURPOSE:** To Calculate the Linear Equation which Comes Closest to Describing a Set of Paired (x,y) Values

- 5.10 NAME:** Prediction Line by Linear Regression.  
**INPUT VARIABLES:** B(I,2) = array of paired values; N = number of pairs.  
**OUTPUT VARIABLES:** B = slope; C = the Y intercept of the prediction equation  $Y = BX + C$ ; R = Pearson correlation coefficient; R2 = coefficient of determination.

```

X00  REM LINEAR REGRESSION
X01  REM INPUT B(I,2), AND N
X02  REM OUTPUT B,C,R, AND R2
X10  LET SX=0
X12  LET SY=0
X14  LET XX=0
X16  LET YY=0
X18  LET XY=0
X20  FOR I=1 TO N
X25  LET SX=SX+B(I,1)
X30  LET SY=SY+B(I,2)
X35  LET XX=XX+B(I,1)**2
X40  LET YY=YY+B(I,2)**2
X50  LET XY=XY+B(I,1)*B(I,2)
X60  NEXT I
X70  LET B=(N*XY-SX*SY)/(N*XX-SX*SX)
X75  LET C=(SY-B**SX)/N
X80  LET R=SQR(B*(N*XY-SX*SY)/
      (N*YY-SY*SY))
X85  LET R2=R*R
X90  RETURN

```

NOTE 1: The value R2, when multiplied by 100, provides an indication of the percent validity (or of accuracy) of the prediction line  $Y = BX + C$ .

NOTE 2: The prediction line is based upon the assumption that the relationship between the paired values is linear.

## Chapter 6

# Business

This chapter contains many useful subroutines, most of which can be used not only in business, but in the home and in school. The simple interest, discount, compound interest, amortization, and sinking fund subroutines are all straightforward and easily used. One caution, however: be certain that interest rates are expressed as a decimal fraction rather than as a percent. For instance, if the interest is  $9\frac{1}{2}$  percent, it must be entered in these subroutines as 0.095.

The Financial Analysis Ratio subroutine (6.11) will make a functional addition to a trial balance or financial statement program. This subroutine could also be added to a program which lists the financial attributes of a number of businesses. Using the subroutine in this way would allow comparisons between companies.

The three depreciation methods (Subroutines 6.12, 6.13, and 6.14) are very useful in that they not only provide a quick calculation of the current year's depreciation of an asset, but provide the asset's book value, after depreciation, as well. As part of an inventory program, one of these subroutines could be used to update the inventory value each year or to estimate the inventory value in some future year. Using all three subroutines in a single program would allow comparisons to be made between the three depreciation methods before a choice of methods is made.




Subroutine 6.15 (TRENDS) is perhaps best used in conjunction with a plotting subroutine from Chapter 9. This sub-

routine makes trends more obvious, because it averages out the misleading peaks and valleys. For instance, if you have 12 monthly sales figures ( $N = 12$ ) and use a 3 month moving average ( $G = 3$ ), this TREND subroutine will provide 10 points ( $P = 10$ ), each of which represents a 3-month average. The larger the number of items grouped into the average (i.e., the larger you make  $G$ ), the more the curve will be smoothed. Data can be input to this subroutine by using the same ARRAY INPUT program provided in Chapter 5 (see page 52).

The remaining subroutines in this chapter (6.16 through 6.20) deal with information storage in a string. This method of information storage may be the most helpful programming tool you will find in this book. Using this method of storage, it is possible to store your data, both alphabetical and numerical, in a single string, or to divide your data into a number of strings (up to 26 strings, A\$ through Z\$).

This string storage method is predicated on groups of information called "substrings." The substrings are linked end-to-end to make up the storage string. For a given storage string, the substrings are all the same length. For example, suppose that we want to store names and telephone numbers. In order to determine the length of the substrings, we must first determine a format for the data. Just for convenience, let us use two initials (allocate 2 spaces for initials), and limit the last name to 12 spaces (that makes a total of 14 spaces for the name). Now, we need 10 spaces for the telephone number (including an area code) and we should insert a space at the end of each substring to separate the substrings from each other.

If we place the last name first (so that it can be easily alphabetized), our format would be as follows:

Substring Format				
Place:	1 . . . 12,	13, 14,	15 . . . 24,	25
				
Use:	Last name	Initials	Numbers	Space
Total Length = 25 spaces = C				

This “total length” which we have chosen, becomes the constant C, which is the number of characters in the substring. C is an input variable of each of the subroutines. Every “group of name and telephone data” placed into a substring and assembled into a storage string must be exactly C characters in length, or the subroutine will not work.

A program to input your data into a substring called X\$ with a length of C (**LEN X\$=C**) might be similar to the following:

```

1  REM "PHONEBOOK"
10 LET I$=""
15 LET C=25
20 PRINT AT 0,7;"INFO_STRING_INPUT";
   AT 3,0;"LAST_NAME_AND
   _SPACES"; AT 5,0;
25 LET X$=""
30 FOR I=1 TO 24
35 INPUT Z$ [or GOSUB to an INKEY$ subroutine]
40 IF Z$=" STOP " THEN STOP
45 IF Z$=" STEP " THEN GOTO 200
50 PRINT Z$;
55 LET X$=X$+Z$
60 IF I=12 THEN PRINT AT 3,0;" _ _ _
   _ _INPUT INITIALS"; AT 5,12;
65 IF I=13 OR I=17 THEN PRINT " _ ";
70 IF I=14 THEN PRINT AT 3,12;"PHONE
   NUMBER"; AT 5,17;
75 IF I=20 THEN PRINT " _ "
80 NEXT I
85 LET X$=X$+" _ "
90 GOSUB Y000 [to storage subroutine such as 6-16]
95 CLS

```



```

100 GOTO 20
200 REM DELETE SUBROUTINE **
210 LET X$=X$(1 TO I-2)
220 LET I=I-2
230 PRINT AT 5,I;"_"
240 PRINT AT 5,0;X$
250 GOTO 80

```

As you can see, this input program is designed to fit the data format we designated for names and phone numbers, and would need to be changed to fit any other desired format. The program also provides a stop (line 40) and a method to delete the last printed character of X\$ (line 45 and lines 200 through 250). Notice, if you replace line 35 with a **GOSUB** to an **INKEY\$** subroutine (see Chapter 8), the task of inputting the data becomes much easier. This same **INKEY\$** subroutine can be used instead of line 60 in the following routine. This **DIRECTORY** routine is simpler and more general than is the **PHONEBOOK** routine:

```

1  REM "DIRECTORY"
10  LET I$=""
20  LET C=25      [this value can be changed to match your
                  substring format length]
30  LET X$=""
40  PRINT AT 0,0;"INFO_STRING INPUT "
50  FOR I=1 TO C-1
60  INPUT Z$      [or GOSUB to an INKEY$ subroutine]
70  IF Z$=" STOP " THEN STOP
80  PRINT Z$;
90  LET X$=X$+Z$
100 NEXT I
110 LET X$=X$+"_"
120 GOSUB Y000    [to storage subroutine]
130 GOTO 30

```

You can build into this input subroutine all the frills you want or need. Too many printing frills may get into the way of usefulness, however. Add only enough frills to make the program easy to use. Start with a simple program adding them one at a time.

The whole purpose of an input program such as **PHONE-BOOK** or **DIRECTORY** is to get your data into the storage subroutines properly, and thus into storage strings. You may store almost any kind of data in these strings: names and test scores, pantry items and costs, even employee numbers, names, wages, dependents, and so on. You can adjust your input and output programs to make this storage system more useful to you.

You need more, however, than just an input program. You must also design an output program which will make the storage system useful to you. If you are calculating payroll, for example, your output program must manipulate the data which is found in your storage system and then print the payroll answers. You may wish to plot data averages, to search for trends, or to calculate and plot the Z scores of a class. If we used **I\$** as a bookkeeping journal, we can print ledger sheets on the monitor by using the substring format given in Subroutine 10.2. In other words, every data input and storage program you design will require an output program as well.

As you can see, the information storage string subroutines can be very useful, but they require that you design input and output programs to meet your specific needs. The input routines given in this chapter can be modified to fit your data format. Output subroutines appear in Chapters 9 and 10.

## SIMPLE INTEREST

**PURPOSE:** To Calculate Maturity Value and Interest Paid on Simple Interest Notes

**6.1 NAME:** Maturity Value at Simple Interest for a Period Given in Days.

**INPUT VARIABLES:** P = principal or face value;  
R = annual interest rate expressed as a decimal value; T = time in days.

**OUTPUT VARIABLES:** MV = maturity value;  
I = interest paid.

```
X00  REM MATURITY VALUE
X01  REM INPUT P, R, AND T
X02  REM OUTPUT MV AND I
X10  LET I=P*R*T/365
X20  LET MV=P+I
X30  RETURN
```

**NOTE:** To change this subroutine to use a time given in months, either multiply the number of months by 30.4 (i.e., for 3 months, let  $T = 3 * 30.4 = 91.2$ ) or change line X10 to read as follows:

```
X10  LET I=P*R*T/12
```

Likewise, if the time T is in years, either multiply the number of years by 365 or change line X10 to read as follows:

```
X10  LET I=P*R*T
```

## 6.2 NAME: Total Simple Interest with Monthly Payments.

**INPUT VARIABLES:** P = principal or face value;  
R = annual rate expressed as a decimal value;  
S = value of monthly payments; N = number of months over which calculation is to be made.

**OUTPUT VARIABLES:** I = total interest due for N months; X = number of payments made; P = outstanding balance after X months; LS = value of last payment (if paid within N months).

```
X00  REM SIMP INT W/PAYMNT
X01  REM INPUT P,R,S, AND N
X02  REM OUTPUT I, P, X AND LS
X10  LET I=0
X15  LET LS=S
```

```

X20  FOR X=1 TO N
X30  LET I=I+P*R/12
X40  IF P<=S THEN GOTO X80
X50  LET P=P+P*R/12-S
X60  NEXT X
X65  LET X=X-1
X70  RETURN
X80  LET LS=(INT(100*(P+P*R/12)+0.5))/100
X90  RETURN

```

## DISCOUNT

**PURPOSE:** To Calculate the Maturity Value, Present Value, and Discount Amount on Discounted Notes

- 6.3 NAME:** Bank Discount of Payable Notes.  
**INPUT VARIABLES:** P = principal or face value of note; R = annual rate charged on note expressed as a decimal value (R = 0 if note is noninterest bearing); T = time of the note expressed in days; R2 = discount rate charged; T2 = discount period in days.  
**OUTPUT VARIABLES:** D = discount on note; MV = maturity value of note; PV = present value (or current value) of note.

```

X00  REM BANK DISCOUNT
X01  REM INPUT P,R,T,R2 AND T2
X02  REM OUTPUT D,MV, AND PV
X10  LET MV=P+P*R*T/360
X20  LET D=MV*R2*T2/360
X30  LET PV=MV-D
X40  RETURN

```

- 6.4** NAME: True Discount of Payable Notes.  
 INPUT VARIABLES: P = principal or face value of note; R = annual rate charged on note expressed as a decimal value (R = 0 if note is noninterest bearing); T = time of the note expressed in days; R2 = discount rate charged; T2 = discount period in days.  
 OUTPUT VARIABLES: D = discount on note; MV = maturity value of note; P = present value (or current value) of note.
- ```

X00  REM TRUE DISCOUNT
X01  REM INPUT P,R,T,R2, AND T2
X02  REM OUTPUT D,MV, AND PV
X10  LET MV=P+P*R*T/365
X20  LET PV=MV/(1+R2*T2/365)
X30  LET D=MV-PV
X40  RETURN
  
```

## COMPOUND INTEREST

**PURPOSE:** To Calculate the Maturity Value or Rate of Interest on Compound Interest Notes

- 6.5** NAME: Maturity Value at Compound Interest.  
 INPUT VARIABLES: P = principal or face value; R = annual rate of interest expressed as a decimal; C = the number of conversion periods per year; N = the number of years (if the number is not an integer, then use decimal fractions).  
 OUTPUT VARIABLES: MV = maturity value of the note; I = total amount of interest paid.
- ```

X00  REM COMPOUND INTEREST
X01  REM INPUT P,R,C, AND N
X02  REM OUTPUT MV AND I
  
```

X10 **LET**  $MV = P * (1 + R/C)^{INT(N*C)}$

X20 **LET**  $I = MV - P$

X30 **RETURN**

- 6.6** NAME: The Compound Rate of Interest, if Maturity Value is Known.

INPUT VARIABLES: P = principal or face value; MV = maturity value; C = number of conversion periods per year; N = the number of years (if the number of years is not an integer, use decimal fractions).

OUTPUT VARIABLES: R = annual rate of interest.

X00 **REM** RATE OF COMP. INT

X01 **REM** INPUT P, MV, C, AND N

X02 **REM** OUTPUT R

X10 **LET**  $R = C * ((MV/P)^{(1/INT(N*C))}) - C$

X20 **RETURN**

## MONTHLY PAYMENTS ON A NOTE (AMORTIZATION)

**PURPOSE:** To Calculate the Number of Payments Needed to Pay an Interest Bearing Note

- 6.7** NAME: Months to Pay Off a Note when Amount of Payment is Fixed.

INPUT VARIABLES: P = principal or face value; S = monthly payment; R = annual rate of interest as a decimal.

OUTPUT VARIABLES: N = number of months to pay off note; LS = last payment amount.

X00 **REM** MONTHS TO PAY OFF

X01 **REM** INPUT P, S, AND R

```

X02  REM OUTPUT N AND LS
X10  LET N=0
X20  LET P=P+P*R/12
X30  LET P=P-S
X40  LET N=N+1
X50  IF P<=S+1 THEN GOTO X70
X60  GOTO X20
X70  LET N=N+1
X80  LET LS=P
X90  RETURN

```

**6.8** NAME: Amount of Payment Required to Pay Note  
when the Number of Payment Months is Known.

INPUT VARIABLES: P = principal or face value;  
N = number of months; R = annual interest rate  
as a decimal.

OUTPUT VARIABLES: S = monthly payment  
required.

```

X00  REM MONTHLY PAYMENT NEEDED
X01  REM INPUT P, N, AND I
X02  REM OUTPUT S AND V
X10  LET R=R/12
X20  LET V=1/(1+R)**N
X30  LET S=P*R/(1-V)
X40  RETURN

```

## SAVINGS (SINKING FUND)

**PURPOSE:** To Calculate the Amount of the  
Monthly Deposit Required to Acquire a Certain  
Value

- 6.9 NAME:** Payment Required to Obtain a Certain Value in a Sinking Fund.

**INPUT VARIABLES:** SF = required end value of sinking fund; N = number of months in which value is to be accumulated; R = annual rate of simple interest paid expressed as a decimal.

**OUTPUT VARIABLES:** D = payment to be deposited each month.

```
X00 REM SINKING FUND PAYMENT
X01 REM INPUT SF,N, AND R
X02 REM OUTPUT D
X10 LET D=SF*R/(12*(1+R/12)**N-12)
X20 RETURN
```

**PURPOSE:** To Calculate the Final Value of a Sinking Fund

- 6.10 NAME:** Future Value of a Sinking Fund

**INPUT VARIABLES:** D = amount to be deposited at the end of each month; N = number of monthly deposits; R = annual rate of simple interest paid expressed as a decimal.

**OUTPUT VARIABLES:** SF = value of the fund at the end of N months.

```
X00 REM FUTURE VALUE
X01 REM INPUT D,N, AND R
X02 REM OUTPUT SF
X10 LET SF=12*D*((1+R/12)**N-1)/R
X20 RETURN
```

## FINANCIAL STATEMENT ANALYSIS RATIOS

**PURPOSE:** To Obtain Ratios Useful in the Analysis of Financial Statements

- 6.11 NAME:** Financial Analysis Ratios.



**INPUT VARIABLES:** NI = net income; CA = current assets; CL = current liabilities; E1 = stockholder's equity at the beginning of the period; E2 = stockholder's equity at the end of the period.

**OUTPUT VARIABLES:** WC = working capital ratio; IE = ratio of net income to average equity.

X00 **REM FINANCIAL ANALYSIS**

X01 **REM INPUT NI,CA,CL,E1,E2**

X02 **REM OUTPUT WC AND IE**

X10 **LET IE=2\*NI/(E1 + E2)**

X20 **LET WC=CA/CL**

X30 **RETURN**

## DEPRECIATION METHODS

**PURPOSE:** To Calculate an Asset's Depreciation for a Given Year of its Useful Life

**6.12 NAME:** Straight Line Depreciation Method.

**INPUT VARIABLES:** C = cost price; S = salvage value; L = life expectancy in years; Y = year of life being depreciated ( $Y \leq L$ ).

**OUTPUT VARIABLES:** DS = depreciation in any year of life; B = book value of asset after depreciation for year Y.

X00 **REM STRAIGHT LINE DEPR**

X01 **REM INPUT C,S,L AND Y**

X02 **REM OUTPUT DS AND B**

X10 **LET DS=(C-S)/L**

X20 **LET B=C-(DS\*Y)**

X30 **RETURN**

**6.13 NAME:** Sum-of-the-Years-Digits Depreciation Method.

INPUT VARIABLES: C = cost price; S = salvage value; L = life expectancy in years; Y = year of life for which depreciation is to be calculated ( $Y \leq L$ ).

OUTPUT VARIABLES: DY = depreciation in year Y; B = book value of asset after depreciation.

```

X00  REM SUM OF YEARS DEPR
X01  REM INPUT C,S,L, AND Y
X02  REM OUTPUT DY AND B
X10  LET X=0
X15  LET B=C
X20  FOR I=1 TO L
X30  LET X=X+I
X40  NEXT I
X50  FOR I=1 TO Y
X60  LET DY=(C-S)*(L-I+1)/X
X70  LET B=B-DY
X80  NEXT I
X90  RETURN

```

#### 6.14 NAME: Double-Declining-Balance Depreciation Method.

INPUT VARIABLES: C = cost price; S = salvage value; L = life expectancy in years; Y = year of life for which depreciation is to be calculated ( $Y \leq L$ ).

OUTPUT VARIABLES: DD = depreciation in year Y; B = book value after depreciation.

```

X00  REM DBL DECLINE BAL
X01  REM INPUT C,S,L, AND Y
X02  REM OUTPUT DD AND B
X10  LET R=1/L
X15  LET B=C

```

```

X20  FOR I=1 TO Y
X30  LET DD=B*R
X40  IF B-DD<=S THEN GOTO X70
X50  LET B=B-DD
X60  NEXT I
X65  RETURN
X70  LET DD=B-S
X80  LET B=S
X90  GOTO X60

```

## TRENDS

**PURPOSE:** To Calculate the Moving Average Trend

**6.15 NAME:** Moving Average Trend.

**INPUT VARIABLES:** D(I) = array of data; N = number of items in D(I); G = number of items to be grouped into moving average.

**OUTPUT VARIABLES:** C(I) = array of moving averages; P = number of averages in the array; MAX = maximum value in array C(I).

```

X00  REM TREND
X01  REM INPUT D(I), N, AND G
X02  REM OUTPUT C(I), P, AND MAX
X10  LET MAX=0
X15  LET P=N-G+1
X20  DIM C(P)
X25  FOR I=1 TO P
X30  LET B=0
X35  FOR J=1 TO G
X40  LET B=B+D(J+I-1)
X50  NEXT J

```

```

X60  LET C(I)=B/G
X70  IF C(I)>MAX THEN LET MAX= C(I)
X80  NEXT I
X90  RETURN

```

## INFORMATION STORAGE

**PURPOSE:** To Store and Manipulate Groups of Similar Types of Information in a Single Storage String

**6.16 NAME:** Information Storage String.

**INPUT VARIABLES:** I\$ = storage string containing groups of information; X\$ = a string containing a single group of information; C = the length of each group contained in I\$, a constant.

**OUTPUT VARIABLES:** I\$ = the storage string including X\$.

```

X00  REM INFORMATION STORAGE
X01  REM INPUT I$,C, AND X$
X02  REM OUTPUT I$
X10  IF LEN X$>C THEN PRINT "INFORMATION
      _STRING_TOO_LONG";X$
X20  IF LEN X$>C THEN RETURN
X30  IF LEN X$=C THEN GOTO X60
X40  LET X$=X$+"_"
X50  GOTO X30
X60  LET I$=I$+X$
X70  RETURN

```

**NOTE 1:** I\$ must have some value before this sub-routine is called. The very first entry into I\$ must be either I\$="" or I\$=X\$ in order to initialize I\$. If no value has been pre-

viously assigned to I\$, line X60 will return an error code of 2/X60.

**NOTE 2:** If  $\text{LEN } X\$ > C$ , it cannot be added to I\$ without upsetting the design of I\$. You may want to print a different message in line X10, and you may want to substitute the command **STOP** for **RETURN** in line X20.

**6.17 NAME:** Delete Information Group in Storage String.

**INPUT VARIABLES:** I\$ = storage string; C = length of information groups in I\$; N = group number of information group to be deleted.

**OUTPUT VARIABLES:** I\$ = storage string with information group N deleted.

```
X00  REM DELETE SUBSTRING
X01  REM INPUT I$, C AND N
X02  REM OUTPUT I$
X10  FOR K=N TO LEN I$/C-1
X20  LET I$(1+(K-1)*C TO K*C)=I$(1+K*C TO
      (K+1)*C)
X30  NEXT K
X40  LET I$(1+LEN I$-C TO LEN I$)=" "
X50  RETURN
```

**6.18 NAME:** Insert Information into Alphabetized Information String.

**INPUT VARIABLES:** I\$ = an alphabetized information storage string; C = length of groups in I\$; X\$ = an information group to be inserted into I\$ ( $\text{LEN } X\$ = C$ ).

**OUTPUT VARIABLES:** I\$ = an alphabetized information string including X\$.

```
X00  REM INSERT ALPHABETIZED
```

```

X01  REM INPUT I$,C,AND X$
X02  REM OUTPUT I$
X10  FOR K=1 TO LEN I$/C
X20  IF I$(1+(K-1)*C TO K*C)>=X$ THEN
      GOTO X40
X25  NEXT K
X30  LET I$=I$+X$
X35  RETURN
X40  IF K=1 THEN LET I$=X$+I$
X45  IF K=1 THEN RETURN
X50  LET I$=I$+I$(1+LEN I$-C TO LEN I$)
X60  FOR J=LEN I$/C-1 TO K STEP -1
X70  LET I$(1+(J-1)*C TO J*C)=I$(1+(J-2)*C
      TO (J-1)*C)
X75  NEXT J
X80  LET I$(1+(K-1)*C TO K*C)=X$
X90  RETURN

```

NOTE 1: This subroutine can be used to assemble an alphabetized list of information groups, or to insert a new piece of information into an existing alphabetized list.

NOTE 2: As with the Information Storage String subroutine (6.16), this subroutine requires that I\$ exists before the subroutine is called (i.e., **LET I\$=""** or **LET I\$=X\$**). Otherwise, the computer will not recognize the symbol I\$, and line X10 will return an error signal of 2/X10.

NOTE 3: To assure that **LEN X\$=C**, you may want to add the following lines to this subroutine:

```

X05  IF LEN X$>C THEN STOP
X06  IF LEN X$=C THEN GOTO X10
X07  LET X$=X$+"_"
X08  GOTO X06

```

NOTE 4: See Chapter 8 for a subroutine which will order an existing list of unordered information groups.

**6.19 NAME:** Retrieval of Storage String Information  
**INPUT VARIABLES:** I\$ = storage string containing groups of information; C = the length of each group contained in I\$, a constant; P = number of the required group in the sequence of groups in I\$.

**OUTPUT VARIABLES:** R\$ = information group stored in position P in storage string I\$.

```

X00  REM INFO RETRIEVAL
X01  REM INPUT I$,C, AND P
X02  REM OUTPUT R$
X10  IF P>LEN I$/C THEN PRINT "GROUP
      NUMBER TOO LARGE"
X20  IF P>LEN I$/C THEN RETURN
X30  LET R$=I$(1+(P-1)*C TO P*C)
X40  RETURN

```

NOTE 1: Although lines X10 and X20 will avoid an error signal if P is too large, you may wish to print a different message in line X10, and may wish to substitute the command **STOP** for **RETURN** in line X20.

NOTE 2: If your program does not allow P to be greater than **LEN I\$/C**, then lines X10 and X20 can be eliminated from the subroutine.

**6.20 NAME:** Search for and Print Item in Storage String.

**INPUT VARIABLES:** I\$ = information storage string; C = length of groups in I\$; S\$ = information string being searched for; L = number of characters to be matched in search ( $L \leq C$ ).

**OUTPUT VARIABLES:** Prints the position number of the found group and prints the found group.

```

X00  REM INFO SEARCH
X01  REM INPUT I$,C,S$, AND L
X02  REM OUTPUT PRINT
X10  FOR J=1 TO LEN I$/C
X20  IF I$(1+(J-1)*C TO L+(J-1)*C)=S$(1 TO
      L) THEN PRINT J;" _";I$(1+(J-1)*C TO
      J*C)
X30  NEXT J
X40  PRINT "END OF SEARCH"
X50  RETURN

```

**NOTE 1:** The choice of the value of L will be determined by your needs. For example, if  $L = 6$  and  $S\$ = \text{"SMITH\_"}'$ , then only those groups beginning with  $\text{"SMITH\_"}'$  will be printed. If  $L = 1$ , however, all groups beginning with the character S will be printed.

**NOTE 2:** Line X40 can be omitted if it is not needed.



## Chapter 7

# Maximum, Minimum, and Sequence

The ability to determine the maximum or minimum value of a disordered array of numbers can be a very helpful start toward bringing order from chaos. As a matter of fact, this chapter might easily have been titled "Order from Chaos." The subroutines described are used for maximums, minimums, averages, sorting, searching, and alphabetizing.

The subroutines in this chapter which deal only with number manipulation require that the input data be in the form of an array. A program for placing your raw data into such an array is given in Chapter 5. (See ARRAY INPUT, p. 52.)

Subroutine 7.1 finds the maximum value in an array. It searches the input data array and sets the variable MX equal to the maximum value found in the array. Subroutine 7.2 works in the same way, but searches for the minimum value in the array and sets the variable MN equal to it. These two search functions are combined in Subroutine 7.3, which finds both the MX value and the MN value, and at the same time finds the arithmetic mean (average) of the array. These three outputs (MAX/MIN/MEAN) provide the range of values and average of the data.

In some instances the actual range of values is not as important as is the range of magnitude of the data. Subroutine 7.4 finds the greatest absolute value in the data, sets MM equal to it, finds the least absolute value (nonzero) in the data, and

sets NM equal to it. If zero is an acceptable value for NM, then line number X40 can be deleted from this subroutine.

In the next two subroutines (7.5 and 7.6), the maximum value of random numbers generated and the repetition of any random number is controlled. Subroutine 7.5 generates random number values which are between M and  $-M$  (where M is an input variable of the subroutine).

Subroutine 7.6 generates positive random integers between 0 and M, and does not allow duplication of any number generated (no two numbers generated will be the same). This subroutine can be most useful when nonrepetitive random numbers are required, such as when the goal is to draw samples from a pool without repeating any sample (as in Bingo). Subroutine 7.6 could also be used to generate a random list of phone numbers for a survey, to list the order in which questions will be placed on a quiz, to give the order in which applicants will be interviewed, or to determine the order in which bills will be paid. It is a very useful subroutine.

Equally useful are the next few subroutines in this chapter, which sort data into order. Subroutine 7.7 is called BUBBLE SORT because the smallest numbers "float" to the top of the list. Subroutine 7.8, called SORT AND SAVE, is identical to BUBBLE SORT except that SORT AND SAVE does not destroy the original order of the array when it sorts the values into ascending order. Instead of sorting the original array, SORT AND SAVE transfers the original array into a new array and sorts the new array. Thus, there is the original array with its order unaltered, and there is a new array containing the original values, which have been sorted into ascending order.

The PARTITION SORT subroutine (7.9) divides the values into two groups. The one group is composed of values which are less than the chosen partition value, and the other group is composed of values which are greater than the chosen partition value. For example, if you have a list of numbers, as shown in column A of Table 7-1 and you choose a partition value of 3, then PARTITION SORT will give you the list shown in column B of Table 7-1.

Table 7-1

A	B
A(1)=9	A(1)=1
A(2)=2	A(2)=2
A(3)=3	A(3)=3
A(4)=7	A(4)=9
A(5)=8	A(5)=7
A(6)=5	A(6)=8
A(7)=6	A(7)=5
A(8)=6	A(8)=6
A(9)=1	A(9)=6

In column B of Table 7-1, A(1) and A(2) are less than the partition value, and A(4) through A(9) are greater than the partition value. The values are not sorted into any order; they simply are partitioned into "greater than" and "less than" groups.

As noted in the listing of the Subroutine 7.9, if the partition value which you choose appears in the data more than once, then the subroutine can become trapped. In order to avoid this possibility, you only need to increase your partition value by some decimal amount, so that the chosen value will not appear on this list. For example, if we wished to use a partition value of 6 with the values in column A, we should use a value of 6.5 (a small decimal value) instead, thus avoiding becoming trapped in an endless loop.

With the Timex/Sinclair string-handling abilities, sorting alphabetical information is almost as easy as sorting numerical data. Subroutine 7.10 will take a list of names or any other alphabetical list in a string, and rearrange the string so that the last name is first. The subroutine does this by beginning at the last letter of the string, and backing up until it finds a space (shown in the program as a "\_"). It then rearranges

the string so that everything between the space and the end of the string (which should be composed of the last name) is removed from the end and placed at the beginning of the string. (Thus the name LAST NAME FIRST.) This subroutine is very useful for converting names to last name first before sorting them or inserting them into alphabetical order. (See Subroutine 6.18 for inserting a string into alphabetical order in an information storage string.)

The last sequence subroutine in this chapter (7.11) is titled ORDER INFO. It will arrange the substrings of an information storage string into ascending order. If the information being stored is alphabetical, then this arrangement of substrings is equivalent to setting the substrings in alphabetical order. If the substring begins with alphabetical information (for example : last name, initials, phone number), the substrings will be alphabetized by the subroutine.

The last two subroutines in this chapter (7.12 and 7.13) will search for a value in an array, and will return the position in the array where the value is found. Subroutine 7.12, which is the most widely used search method, will search any array. It starts at the beginning, A(1), and checks each value in the array, in order, until it finds a match. It then returns P as the position of the match, meaning that the match was found at A(P). If no match is found, then  $P = 0$ .

Subroutine 7.13 is a binary rather than a sequential search. A binary search is much faster than the sequential search, but requires that the values of the searched array be in ascending order. Subroutine 7.13 works by dividing the given list in half and jumping to the middle of the list. If the value found in the middle is greater than the value searched for, the subroutine then jumps to the middle of the half of the list containing the smaller values. If instead the first value found is smaller than the one searched for, the subroutine jumps to the middle of the half containing the larger values. The subroutine continues to jump to the middle of the dwindling group of halved lists, always jumping in the direction of the searched-for value.

Searching for alphabetical data is covered by Subroutines 6.19 and 6.20, in Chapter 6. These subroutines will allow you to find alphabetical information either by name or by list position number.

**MAXIMUM AND MINIMUM VALUES**

**PURPOSE:** To Find the Maximum Value of a Set of Values

- 7.1**    **NAME:** Maximum Value in an Array of Numbers.  
**INPUT VARIABLES:** A(I) = array of values; N = length of array (number of values in the array).  
**OUTPUT VARIABLES:** MX = the maximum value stored in the array.

```

X00  REM MAXIMUM VALUE
X01  REM INPUT A(I) AND N
X02  REM OUTPUT MX
X10  LET MX=A(1)
X20  FOR I=2 TO N
X30  IF A(I)>MX THEN LET MX=A(I)
X40  NEXT I
X50  RETURN

```

**PURPOSE:** To Find the Minimum Value of a Set of Values

- 7.2**    **NAME:** Minimum Value in an Array of Numbers.  
**INPUT VARIABLES:** A(I) = array of values; N = length of array (number of values in the array).  
**OUTPUT VARIABLES:** MN = minimum value.

```

X00  REM MINIMUM VALUE
X01  REM INPUT A(I) AND N
X02  REM OUTPUT MN
X10  LET MN=A(1)
X20  FOR I=2 TO N
X30  IF A(I)<MN THEN LET MN=A(I)
X40  NEXT I
X50  RETURN

```

**MAXIMUM, MINIMUM, MEAN (AVERAGE)**

**PURPOSE:** To Find the Maximum, Minimum, and Average Value of a Set of Values

**7.3 NAME:** Maximum, Minimum, and Mean.

**INPUT VARIABLES:** A(I) = array of values; N = number of values in array.

**OUTPUT VARIABLES:** MX = maximum; MN = minimum; AV = average.

```

X00  REM MAX, MIN, MEAN
X01  REM INPUT A(I) AND N
X02  REM OUTPUT MX, MN, AND AV
X10  LET MX=A(1)
X15  LET MN=A(1)
X20  LET S=0
X30  FOR I=1 TO N
X40  LET S=S+A(I)
X50  IF A(I)>MX THEN LET MX=A(I)
X60  IF A(I)<MN THEN LET MN=A(I)
X70  NEXT I
X80  LET AV=S/N
X90  RETURN

```

**NOTE 1:** To find which number in an array is maximum or minimum, add lines X45 and X55 as shown:

```

X45  IF A(I)>MX THEN LET X=I
X55  IF A(I)<MN THEN LET R=I

```

X will be the position in the array of the value MX, and R will be the position in the array of the value MN. If there is more than one array value which equals MX, then only the first one will be identified by X.

Likewise, R will identify only the first value equal to MN.

NOTE 2: If you want to limit the value AV to two decimal places, you can do it by changing line X80 to read as shown below. The +0.5 value will cause rounding to the nearest decimal. Without this +0.5 value, the computer will round only downward.

X80    **LET** AV=(INT(S\*100/N+0.5))/100

NOTE 3: If average value AV is not needed, then lines X20, X40, and X80 may be deleted.

## MAXIMUM AND MINIMUM MAGNITUDE (NONZERO)

**PURPOSE:** To Determine the Largest and Smallest (Nonzero) Magnitude in an Array of Numbers

**7.4 NAME:** Maximum and Minimum Magnitude.

**INPUT VARIABLES:** A(I) = array of values; N = number of values in the array.

**OUTPUT VARIABLES:** MM = maximum magnitude; NM = minimum magnitude.

```
X00  REM MAX AND MIN MAG
X01  REM INPUT A(I),N
X02  REM OUTPUT MM AND NM
X10  LET MM=0
X20  LET NM=ABS A(1)
X30  FOR I=1 TO N
X40  IF A(I)=0 THEN GOTO X70
X50  IF ABS A(I)>MM THEN LET MM=ABS A(I)
X60  IF ABS A(I)<NM THEN LET NM=ABS A(I)
X70  NEXT I
X80  RETURN
```

## RANDOM NUMBERS

**PURPOSE:** To Control the Degree of Randomness of Generated Random Numbers

**7.5 NAME:** Range of Random Numbers.

**INPUT VARIABLES:** M = maximum value of random number required.

**OUTPUT VARIABLES:** R = random number generated.

X00 **REM** RANDOM NUMBER RANGE

X01 **REM** INPUT M

X02 **REM** OUTPUT R

X05 **RAND**

X10 **LET** R=INT(M\*RND)+1

X20 **LET** S=INT(2\*RND)

X30 **IF NOT** S **THEN LET** R=-R

X40 **RETURN**

**NOTE:** If only positive numbers are required, then omit lines X20 and X30.

**7.6 NAME:** Unique (Nonrepeating) Random Positive Number Generator.

**INPUT VARIABLES:** M = maximum value; N = number of values needed.

**OUTPUT VARIABLES:** R(I) = array of nonrepeating random numbers.

X00 **REM** UNIQUE RANDOM NUMBERS

X01 **REM** INPUT M **AND** N

X02 **REM** OUTPUT R(I)

X05 **RAND**

X10 **DIM** R(N)

X20 **FOR** I=1 **TO** N

X30 **LET** R(I)=1+INT(M\*RND)



```

X40  IF I=1 THEN GOTO X80
X50  FOR J=1 TO I-1
X60  IF R(J)=R(I) THEN GOTO X30
X70  NEXT J
X80  NEXT I
X90  RETURN
    
```

## SORT NUMBERS

**PURPOSE:** To Sort a Series of Numbers into  
Ascending Order

- 7.7**    **NAME:** Bubble Sort.  
**INPUT VARIABLES:** A(I) = list of values; N =  
                           number of values in list.  
**OUTPUT VARIABLES:** A(J) = ordered list of  
                           values.

```

X00  REM BUBBLE SORT
X01  REM INPUT A(I) AND N
X02  REM OUTPUT A(J)
X10  FOR I=1 TO N-1
X20  FOR J=I TO N
X30  IF A(J)>A(I) THEN GOTO X70
X40  LET X=A(I)
X50  LET A(I)=A(J)
X60  LET A(J)=X
X70  NEXT J
X80  NEXT I
X90  RETURN
    
```

- 7.8**    **NAME:** Bubble Sort and Save.  
**INPUT VARIABLES:** A(I) = list of values; N =  
                           number of values in list.

**OUTPUT VARIABLES:** A(I) = unaltered list; B(I)  
= ordered list of values.

```

X00  REM SORT AND SAVE
X01  REM INPUT A(I) AND N
X02  REM OUTPUT A(I) AND B(I)
X10  DIM B(N)
X15  LET X=0
X20  FOR I=1 TO N-1
X30  FOR J=I TO N
X40  IF X=0 THEN LET B(J)=A(J)
X50  IF B(J)>B(I) THEN GOTO X75
X60  LET Y=B(I)
X65  LET B(I)=B(J)
X70  LET B(J)=Y
X75  NEXT J
X80  LET X=1
X85  NEXT I
X90  RETURN

```

**PURPOSE:** To Divide a List of Numbers into  
those Numbers Less than and those Numbers  
Greater than a Given Value

### 7.9 NAME: Partition Sort.

**INPUT VARIABLES:** A(I) = array of numbers; N  
= number of values in array; L = partition value  
(value used to divide array).

**OUTPUT VARIABLES:** A(J) = sorted array.

```

X00  REM PARTITION SORT
X01  REM INPUT A(I),N, AND L
X02  REM OUTPUT A(J)
X10  LET J=N
X20  FOR I=1 TO J-1

```

```

X25  IF A(I) >= L THEN GOTO X40
X30  NEXT I
X35  RETURN
X40  FOR J=N TO I+1 STEP -1
X45  IF A(J) <= L THEN GOTO X60
X50  NEXT J
X55  RETURN
X60  LET Y=A(I)
X65  LET A(I)=A(J)
X70  LET A(J)=Y
X80  IF J>I+1 THEN GOTO X20
X90  RETURN

```

NOTE: If there is more than one number in A(I) whose value equals the value of L, this sub-routine may become trapped in an endless loop. To avoid this possibility, simply increase L to a decimal value which cannot be found in A(I). For example: If A(I) is made up of whole numbers and L is 22, simply increase L to 22.5. In this way no value of A(I) can equal L, and yet your partition will be at the same place in the array.

## LAST NAME FIRST

**PURPOSE:** To Convert a Name in A\$ to the Form of Last Name First

**7.10 NAME:** Last Name First.

**INPUT VARIABLES:** A\$ = name (last name last).

**OUTPUT VARIABLES:** B\$ = name (last name first).

```

X00  REM LAST NAME FIRST
X01  REM INPUT A$
X02  REM OUTPUT B$

```

```

X10  FOR I=LEN A$ TO 1 STEP -1
X20  IF A$(I)="_" THEN GOTO X40
X30  NEXT I
X40  LET B$=A$(I+1 TO LEN A$)+","_"+A$
      (1 TO I-1)
X50  RETURN

```

## ALPHABETIZING AN INFORMATION STRING

**PURPOSE:** To Place Information Substrings into an Ascending Order

- 7.11 NAME:** Order Information Substrings.  
**INPUT VARIABLES:** I\$ = an information storage string containing groups of information; C = the length of each information group in I\$.  
**OUTPUT VARIABLES:** I\$ = an information storage string containing groups of information in an ascending alphabetic/numeric order.

```

X00  REM ORDER INFO
X01  REM INPUT I$ AND C
X02  REM OUTPUT I$
X10  FOR I=1 TO LEN I$/C-1
X20  FOR J=1 TO LEN I$/C
X30  IF I$(1+(J-1)*C TO J*C)>I$(1+(I-1)*C TO I*C) THEN GOTO X70
X40  LET X$=I$(1+(I-1)*C TO I*C)
X50  LET I$(1+(I-1)*C TO I*C)=I$(1+(J-1)*C TO J*C)
X60  LET I$(1+(J-1)*C TO J*C)=X$
X70  NEXT J
X80  NEXT I
X90  RETURN

```

## SEARCH FOR NUMBERS

**PURPOSE:** To Find the Position of a Required Number in a Table of Numbers

**7.12 NAME:** Sequential Search.

**INPUT VARIABLES:** A(I) = array of values; N = number of values in the array; S = value searched for.

**OUTPUT VARIABLES:** P = position of S in array (0 if not found).

```

X00  REM SEQUENTIAL SEARCH
X01  REM INPUT A(I),N, AND S
X02  REM OUTPUT P
X10  LET P=0
X20  FOR I=1 TO N
X30  IF S<>A(I) THEN GOTO X60
X40  LET P=I
X50  GOTO X70
X60  NEXT I
X70  RETURN

```

**NOTE:** This subroutine will search an array which is in any order. It is very useful when N is not large, and is the only subroutine to use when A(I) is unordered.

**7.13 NAME:** Binary Search.

**INPUT VARIABLES:** A(I) = ordered array of values; N = number of values in array; S = value searched for.

**OUTPUT VARIABLES:** P = position of S in array (0 if not found).

```

X00  REM BINARY SEARCH
X01  REM INPUT A(I),N, AND S
X02  REM OUTPUT P

```

```
X03  REM A(I) MUST BE ORDERED
X10  LET T=0
X15  LET B=N
X20  LET V=INT((T+B+1)/2)
X25  IF S=A(V) THEN GOTO X80
X30  IF V=B OR V=T THEN GOTO X65
X40  IF S<A(V) THEN GOTO X55
X45  LET T=V
X50  GOTO X20
X55  LET B=V
X60  GOTO X20
X65  LET P=0
X70  GOTO X90
X80  LET P=V
X90  RETURN
```

NOTE: This search subroutine is rapid, but will work only on a list which is already sequenced in ascending order.

## INKEY\$ and SCROLL

**INKEY\$** and **SCROLL** are very helpful functions in the Timex/Sinclair 1000 and Sinclair ZX computers. When properly manipulated, they allow you to design programs which are more “user friendly.” The same is true of the **INKEY\$** function in the Timex/Sinclair 2000 and the Sinclair Spectrum. However, because these color computers do not have a **SCROLL** function that is completely operator dependent, the **SCROLL** subroutines in this chapter apply only to the Timex 1000 and the Sinclair ZX Computers.

Because operator control of the **SCROLL** function can be very useful, a separate discussion of the use of this function for the Timex/Sinclair 2000 and the Sinclair Spectrum is included later in this chapter.

Although it is possible to write useful programs without using the **INKEY\$** or **SCROLL** function, these functions can be most helpful, especially if your program requires inputting or printing of data. The two functions **INKEY\$** and **SCROLL** can be applied in ways which will cause your programs to be much easier to use.

The **INKEY\$** function scans the Timex/Sinclair keyboard and reports the character of the depressed key. This function will report any regular or “shifted” keyboard character. It will not, however, report keyboard actions, such as “delete”, “graphics”, or the direction arrows, and it will not accept a space. (If **INKEY\$** is asked to report a space, it will instead

report error code D and stop the program.) In spite of this drawback, **INKEY\$** can be used for almost any input situation, and if required the space report drawback can be circumvented.

Subroutine 8.1 is an **INKEY\$** subroutine for numeric input only. This subroutine places the variable X equal to the value of the number which **INKEY\$** reports. Because **INKEY\$** is a string function, it must be treated as a string. Thus:

```
X20  LET X=VAL INKEY$
```

Line X20 is necessary in order to get **INKEY\$** into the form of a numeric variable. The **PAUSE 9000** instruction of line X10 in the subroutine provides 150 seconds of delay in which a key may be pressed. If no key is pressed by the time the **PAUSE** is complete, then line X20 sets  $X = 0$  and the subroutine returns to the main program. The **PAUSE** function is used so that the program can be run in the **FAST** mode (see **PAUSE** and **INKEY\$** in your computer manual).

To use **INKEY\$** in the **SLOW** mode, or to use **INKEY\$** in the Timex/Sinclair 2000 or Sinclair Spectrum, two modifications must be made. First, the **PAUSE** line must be changed to read:

```
XX0  IF INKEY$ < > "" THEN GOTO XX0
```

Then, the **POKE** line which follows the **PAUSE** line must be changed to read:

```
XX5  IF INKEY$ = "" THEN GOTO XX5
```

These changes apply to Subroutines 8.1 through 8.4.

Cursor **INKEY\$** (Subroutine 8.2) is a printing subroutine. The print position (line and column) is provided as two input variables. First, a "cursor" is printed at the designated print position to indicate on the screen where the next character will be printed. Then, when the keyboard is touched, the **INKEY\$** character is printed in place of the cursor. If you want to save the value of the **INKEY\$**, line X25 can be added as shown:

```
X25  LET A$=INKEY$
```



A typical example of how Subroutine 8.2 can be used is shown in the following program:

```

1  REM "TELE-NO"
10  FOR I=1 TO 15
20  FOR J=1 TO 10
30  LET L=I
40  LET C=J
50  IF C>3 THEN LET C=C+1    [provides space
                               after area code]
60  IF C>7 THEN LET C=C+1    [provides space
                               between first
                               three and last
                               four numbers]

70  GOSUB Y000    [to Subroutine 8.2, CURSOR INKEY$]
80  NEXT J
90  NEXT I
100 STOP

```

The TELE-NO program will accept 15 telephone numbers and print them on the screen. Line X50 places a space between the area code and the local number. Line X60 places another space between the first three digits and the last four digits of the local number. The CURSOR INKEY\$ subroutine provides a visual prompt, in the form of the cursor, as to where the next number input will be printed.

This example program may be a trivial use of the Subroutine 8.2, but the program illustrates how this subroutine might be used.

The next subroutine, ALPHA INKEY\$ (8.3), is simply an embellishment of the preceding INKEY\$ subroutines. Here, line X20 sets A\$ equal to INKEY\$, and line X30 changes A\$ to a space ( ) if INKEY\$ is ">". This change in A\$ simply means that because INKEY\$ will not accept a space as an input, we have chosen the ">" as the space bar. (Any other acceptable character could have been chosen as the substitute for the space.) If no decimals or periods are needed in your

printing, then the period (.) might be an excellent substitute for the space bar. Simply change line X30 to read:

```
X30  IF A$="." THEN LET A$="_"
```

The last **INKEY\$** subroutine (8.4) has some word-processing attributes. This subroutine will print up to 22 lines of 32 characters and will store this "page" of characters in string A\$.

IF **STOP** is pressed or if the screen is filled, then Subroutine 8.4 will return to the main program. A **BACKSPACE** AND **DELETE** subroutine could easily be added (see **PHONEBOOK** in Chapter 6, lines 200–250), which would allow you to correct any mistakes.

Notice that parts of Subroutine 8.4 are taken from Subroutine 8.3 (see lines X30, X40, X50). You could delete lines X40 and X50 and change line X30 to:

```
X30  GOSUB Y00 [to Subroutine 8.3]
```

Making this change will not affect the way Subroutine 8.4 works, but you must change the output variable of Subroutine 8.3 from A\$ to B\$ in order to match the input variables in Subroutine 8.4.

### T/S 1000 and ZX81 SCROLL

Like the **INKEY\$** function, the **SCROLL** function is quite helpful at times, but by itself seldom is adequate. If we are putting data into a table and the screen is filling up, we must keep an eye on the bottom line so that we can **SCROLL** before we get a full screen and a 5 error signal. Without constantly counting how many lines have been printed, it is often difficult to be certain which line is the bottom line of the screen. Moreover, if we do **SCROLL** in time, the table heading goes off the top of the screen. In such a case, it may be hard to remember whether column 3 was the sales volume for 1984 or was the inventory of washroom keys. The **SCROLL** subroutines in this chapter can be used to overcome many such problems.

The first **SCROLL** subroutine (8.5) allows you to scroll by simply pressing ENTER. Initially, Subroutine 8.5 prints a prompt on the bottom line of the screen (line X10). If anything other than ENTER is input, the subroutine erases the prompt (line X60) and returns to the main program.

**AUTO-SCROLL** (Subroutine 8.6) is very practical and can be combined with other subroutines to give even more benefit. Line X10 looks into the computer and determines if the last line has been printed. If it has, the subroutine scrolls, and line X20 tells the computer that it can now print on the last line. This subroutine is truly an automatic scroll. Using the subroutine will allow the computer to check for a full screen and then to scroll, if necessary, before it prints.

The automatic function of Subroutine 8.6 can be expanded by changing line X10 to read:

```
X10  IF PEEK 16442=2 THEN GOSUB Y00
```

If you were printing a table, the **GOSUB** might take you to Subroutine 8.7, which will preserve a table heading while scrolling the data in the table. This subroutine does so by looking into the computer (**PEEK**), by taking the 33 characters which the computer is displaying on line 0 (actually 32 characters and an "end-of-line" sign), and by storing the character's code in an array A(X). The subroutine now scrolls (line 50). Scrolling rolls line 0 off the top of the screen, but we have already stored it in array A(X). We can now tell the computer (with **POKE**) to use the array A(X) as the codes for printing line 0.

The functions **PEEK** and **POKE** are operations for looking into and altering the computer's memory. In Subroutine 8.7 we have **PEEK**ed into the memory, stored what we found, and later **POKE**d that stored information back into the same place we found it. This whole operation could be accomplished much faster by using machine code (the language of the computer), but to do this, as we will see in Chapter 10, is much more complicated.

Subroutine 8.8 works in a manner similar to 8.7, but preserves the footing of a table rather than the heading. By mak-

ing changes indicated in the notes of the subroutines, it is possible to save two lines of heading and two lines of footing while scrolling the 18 lines of body copy which fall in between.

It should be noted that the **AUTO-SCROLL** subroutine (8.6) cannot be used in conjunction with the **SCROLL WITH FOOT** Subroutine (8.8). This is because **AUTO-SCROLL** will always see the footing printed on line 21 and will assume that the screen is filled. You may find that with changes in Subroutine 8.6 that you can circumvent this limitation.

### T/S 2000 and Spectrum SCROLL

The Timex/Sinclair color computers provide for an automatic visual scroll prompt when the screen is filled. This provision does exactly the same thing that Subroutine 8.5 does, except that it does it without using print space. Subroutine 8.5 uses line 21 to tell you how to **SCROLL**, while the color computer prints the question **SCROLL ?** *below* line 21 on the screen. Pressing any key but **STOP**, **BREAK**, or **N** will cause the screen to scroll up one line. In some cases, the computer will operate identically to the **AUTO-SCROLL** in Subroutine 8.6.

Because of the built-in **SCROLL** function in the color computers, Subroutines 8.7 and 8.8 cannot be used with them. Additionally, the methods used by the color computers to store the display data is much more complex than is the storage in the other Timex/Sinclair computers. For these reasons, if you use the color computers you should replace Subroutines 8.7 and 8.8 with subroutines which **PRINT** a head or foot on the screen following a **SCROLL**.

The system variable **SCR-CT** on the color computers can be used to control the **SCROLL** function by **POKE**ing 1 or 2 into the memory address 23692. **POKE**ing 1 will hold the **SCROLL** for manual operation, while **POKE**ing 2 will cause one line of **SCROLL** (3 will **SCROLL** twice, 4 three times, etc.). Additionally, the line number address of the system variable **S-POSN** can be **POKE**d to fool the color computers into not **SCROLL**ing, or can be **PEEK**d to determine when to **SCROLL**. The **S-POSN** address (23689) contains the line

number of the **PRINT** position. A **PRINT** position line number of 24 indicates the top line on the screen, and 3 is the bottom line of the printing area. Thus, if you get the answer 3 when you **PEEK** 23689, then the computer will print next on the bottom line (line 21) of the print area.

### Conclusion

All of the subroutines in this chapter can help you to make your programs more "user friendly." More importantly, however, they present interesting ideas and interesting directions to explore. If you are not familiar with the **PEEK** and **POKE** operations, you may wish to read about them in the section titled "System Variables" of your computer manual. There you will find that Subroutine 8.6 looks into the system variable called S-POSN to find the line number of the next print position. You will also find that Subroutines 8.7 and 8.8 look into the D-FILE system variable to read what is being displayed on the screen. You do not need this information in order to use the subroutines, but as you learn more about how the computer works, you can cause the computer to do exactly what you want, pronto. And that can be exciting!

## INKEY\$

**PURPOSE:** To Provide the Value of a Depressed Key

**8.1 NAME:** Auto **INKEY\$** Value.

**INPUT VARIABLES:** None (depressed key).

**OUTPUT VARIABLES:** X = numerical value of depressed key.

X00 **REM INKEY\$** VALUE

X01 **REM INPUT** KEY PRESS

X02 **REM OUTPUT** X

X10 **PAUSE** 90000

X15 **POKE** 16437,255

X20 **LET X=VAL INKEY\$**

X30 **RETURN**

**NOTE:** This subroutine is useful only for inputting numeric information. For alphabetic or alphanumeric information, see Subroutines 8.3 and 8.4.

**PURPOSE:** To Print **INKEY\$** at a Point Indicated by a Cursor

**8.2 NAME:** Cursor **INKEY\$**.

**INPUT VARIABLES:** L = print line number; C = print column number.

**OUTPUT VARIABLES:** None (print at cursor position).

X00 **REM CURSOR INKEY\$**

X01 **REM INPUT L AND C**

X02 **REM OUTPUT PRINT**

X10 **PRINT AT L,C;"█"**

X20 **PAUSE 9000**

X25 **POKE 16437,255**

X30 **PRINT AT L,C;INKEY\$**

X40 **RETURN**

**NOTE:** The subroutine will not accept a "space" as an **INKEY\$** input. However, this limitation can be overcome, as shown in the next subroutine (8.3).

**PURPOSE:** To Expand the Flexibility of the **INKEY\$** Function

**8.3 NAME:** **INKEY\$** with Spaces.

**INPUT VARIABLES:** None (key depression).

**OUTPUT VARIABLES:** A\$ = **INKEY\$** or space.

X00 **REM ALPHA INKEY\$**

X01 **REM INPUT INKEY\$**

```

X02  REM OUTPUT A$
X10  PAUSE 9000
X15  POKE 16437,255
X20  LET A$=INKEY$
X30  IF A$=" ">" THEN LET A$="_"
X40  RETURN

```

NOTE: Because INKEY\$ cannot accept a space as an input, this subroutine has been adapted to provide a space when SHIFT and M are depressed together (this shifted-M would normally produce the > symbol). Depressing SPACE in this subroutine will cause a D error code.

#### 8.4 NAME: INKEY\$ Print and Save.

INPUT VARIABLES: None (key depression).

OUTPUT VARIABLES: A\$ = string of printed content.

```

X00  REM INKEY$ PRINT AND SAVE
X01  REM INPUT INKEY$
X02  REM OUTPUT A$
X10  LET A$=""
X15  FOR I=0 TO 21
X20  FOR J=0 TO 31
X30  PAUSE 9000
X35  POKE 16437,255
X40  LET B$=INKEY$
X45  IF B$="STOP" THEN RETURN
X50  IF B$=" ">" THEN LET B$="_"
X60  LET A$=A$+B$

```

```

X70 PRINT AT I,J;B$
X80 NEXT J
X85 NEXT I
X90 RETURN

```

NOTE: Control will return to the main program either when the screen is filled or when "SHIFT" and A are depressed (STOP).

## SCROLL

**PURPOSE:** To Provide Controlled Use of the **SCROLL** function

**8.5 NAME:** Enter to **SCROLL**.

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```

X00 REM ENTER TO SCROLL
X10 PRINT AT 21,0;" ___PRESS_"ENTER"
    TO SCROLL_____
X20 INPUT Z$
X30 IF Z$<>" THEN GOTO X60
X40 SCROLL
X50 PRINT AT 20,3;" ___[23 spaces]___"
X60 PRINT AT 21,3;" ___[23 spaces]___"
X70 RETURN

```

**8.6 NAME:** Automatic **SCROLL** at Line 21.

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```

X00 REM AUTO-SCROLL
X10 IF PEEK 16442=2 THEN SCROLL

```



```

X20  IF PEEK 16442=2 THEN POKE 16442,3
X30  RETURN

```

NOTE: This subroutine checks to see if another line can be printed on the screen. If not, it scrolls and sets up to print on line 21.

**PURPOSE:** To Scroll Body Copy while Retaining the Headings of a Chart or Table

**8.7 NAME:** **SCROLL** with Heading.

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```

X00  REM SCROLL WITH HEAD
X10  DIM A(33)
X20  FOR X=1 TO 33
X30  LET A(X)=PEEK((PEEK 16396+256*PEEK
      16397)+X)
X40  NEXT X
X50  SCROLL
X60  FOR X=1 TO 33
X70  POKE((PEEK 16396+256*PEEK
      16397)+X),A(X)
X80  NEXT X
X90  RETURN

```

NOTE 1: To save both print lines 0 and 1, change all 33s to 66s in lines X10, X20 and X60.

NOTE 2: If the heading has been previously stored in A(X) [the subroutine has been used previously to store the required heading], **GO-SUB X50** will provide the same results more quickly than will **GOSUB X00**.

**PURPOSE:** To Scroll Body Copy while retaining the Footings of a Chart or Table

**8.8 NAME:** **SCROLL** with Footing.

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```

X00  REM SCROLL with FOOT
X10  DIM B(33)
X20  FOR X=1 TO 33
X30  LET B(X)=PEEK((PEEK 16396+256*PEEK
      16397)+693+X)
X40  NEXT X
X45  PRINT AT 20,0;" ____ [32 spaces] ____"
X50  SCROLL
X60  FOR X=1 TO 33
X70  POKE((PEEK 16396+256*PEEK
      16397)+693+X),B(X)
X80  NEXT X
X90  RETURN

```

**NOTE 1:** To save both print lines 20 and 21, change all 33s to 66s in lines X10, X20, and X60. Change the number 693 to 660 in lines X30 and X70, and change the 20 in program line X45 to 19.

**NOTE 2:** If the required footing has been stored in B(X) by previous use of the subroutine, then **GOSUB X45** will provide the same results more quickly than will **GOSUB X00**.

NOTE 3: Both footings and headings can be stored by combining Subroutines 8.7 and 8.8. To combine these subroutines, take lines X10, X30, and X70 from the Subroutine 8.7 and change the line numbers to X15, X35, and X75. Then insert these lines into Subroutine 8.8.

## Chapter 9

# Graphing

Most of the subroutines in this book deal with the input and manipulation of data. In contrast to the other chapters, Chapters 9 and 10 are devoted to methods for the *output* of your data. It is assumed that you will devise output programs which will present your data in a manner useful to you.

The graphing subroutines in Chapter 9 can be added to almost any data manipulation routine to illustrate, to compare, or simply to present the data. With the exception of the co-ordinate system plotting subroutines (9.7 and 9.8), all of the subroutines in this chapter have both a **PRINT** form and a **PLOT** form. Each form has both advantages and disadvantages.

The **PLOT** form has the advantage of using a pixel, which is only one-quarter the size of a print position. This pixel size means the plotting screen is 64 pixels wide and 44 pixels high. Such an increase in plotting points provides much more space for data (if needed), and plots a smoother curve than can a **PRINT** graph. However, the **PLOT** form uses only one character, which is the plot-pixel. This plot-pixel is simply a small black square, one-quarter the size of a print position space. All of the **PRINT** forms of these graphing subroutines can be used as shown for all of the Timex and Sinclair Computers. The **PLOT** subroutines, however, must be modified to be used on the Timex/Sinclair 2000 or the Sinclair Spectrum. These two color computers use a much smaller plotting pixel, so a dimension conversion is required. Multiplying each of the **PLOT** coordinates in this chapter by 4 will usually cause the

**PLOT** subroutines to work properly with the Timex 2000 and the Sinclair Spectrum. (This conversion is not difficult, but may be tedious.) Additional information about using the **PLOT** subroutines for the T/S 2000 and the Spectrum is given later in this chapter.

In contrast the **PRINT** form has only the normal 22 lines of 32 characters on which to print (or a screen 32 units wide and 22 units high on which to graphically represent points.) Although this smaller number of graph points may be limiting for some data, the **PRINT** form has the advantage of being able to display a variety of characters. For instance, when using the **PRINT** form, you can plot one set of data using the character \*, while plotting another set (on the same graph) using the character +. When printing a pie graph, each segment can use a different graphic symbol (■, ▒, ▓, etc.) or a different identifying letter or character (A, B, S, >, etc.). Graphing in the **PRINT** form can be impressive, if it allows enough detail for your data.

The first two subroutines in this chapter illustrate the difference between the two graphing methods. The first subroutine (9.1) prints a dot (.) at the required graphing point. The printed character (which in this case is a period printed by line X30) can be changed to a +, an X, a \*, or any other print character desired. This subroutine cannot, however, print more than 30 points, and line X10 limits the input to the first 30 points of the input data array. If your data has less than 31 points, then this limit is of no consequence.

In contrast, the second subroutine (9.2) will accept and plot up to 63 data points, but the plotting symbol cannot be changed. The plotting symbol for Subroutine 9.2 must always be the small pixel square provided by the **PLOT** function of the computer.

Both Subroutines 9.1 and 9.2 will accept any positive maximum data value (MAX) and proportionately scale all of the plotted data to fit onto the monitor screen. The value of MAX is printed in the upper-left-hand corner of the screen to provide the scale for the graph. The horizontal scale is also adjusted to fill the screen. When less than the maximum possible number of data points are input, the subroutines will distribute the points proportionately over the screen width.

All of the subroutines in this chapter will automatically adjust the height and width of the graph to fit the **PRINT** or **PLOT** parameters of the monitor screen. They will also limit the maximum number of data points which can be plotted. (This maximum will be determined by the format used.)

Subroutine 9.3 will print a bar graph, using your choice of any print characters. The code for the character which you choose is input as the input variable A. This code for the different characters can be found in your computer instructional manual in the "Character Set" appendix. Using the character code 38 will cause the bars of the graph to be constructed of A's, while a code of 8 would print bars of █s.

Subroutine 9.4 plots only solid black bars, but can plot twice as many bars on the screen as can 9.3. Notice also that the accuracy of the plotting is better with a **PLOT** subroutine such as 9.4 than with a **PRINT** subroutine such as 9.3. This is true simply because the plotting pixel is half the width of the printing space.

The two horizontal bar graph subroutines (9.5 and 9.6) are normal bar graphs turned on their sides. Because the monitor screen is wider than it is high, the horizontal bar graphs can print fewer bars than can the normal graph, but the bars can be longer (which improves the accuracy of the graph).

The next four subroutines (9.7a, 9.7b, 9.8a, and 9.8b) are in pairs. The "A" half of the pair (9.7a and 9.8a) will print a coordinate system. The "B" half of the pair will plot points on that coordinate system. Subroutine 9.7a prints a four-quadrant cartesian system. Subroutine 9.7b takes negative or positive  $x$  and  $y$  coordinates as input and plots on the graph system the point described by the coordinates.

Subroutines 9.8a and 9.8b do exactly the same thing with a polar coordinate system. Subroutine 9.8a prints the polar system, and 9.8b plots points on the system. Both "b" subroutines will **RETURN** if the input data is not properly dimensioned to plot on the screen.

The FROM-TO subroutine (9.9) will plot a line between two sets of cartesian ( $x,y$ ) points. This subroutine accepts only positive values of  $x$  and  $y$ , and the "origin" of the graph (point 0,0) is in the lower, left-hand corner of the screen. Instructions

are given for changing the coordinate arrangement into a four-quadrant system with the point 0,0 appearing in the middle of the screen. Subroutine 9.7a can then be used to print a graph system for plotting FROM-TO.

The most interesting graph methods may be the pie graph subroutines. Subroutine 9.10 does nothing but plot a circle in which the pie segments may later be printed or plotted. Subroutine 9.11 plots a circle to outline the pie and plots one segment of the pie graph. This subroutine is useful for two-part graphs, or can be used with the next subroutine (9.12) for multiple-part graphs. The segment plotted by 9.11 is solid black, but 9.12 will plot solid or checked segments, depending upon the value of the input variable X. The choice of plotting solid, of plotting in checks, or of not plotting gives three distinct segments. The following program will illustrate these distinction methods by plotting nine 40-degree pie segments with alternating patterns.

```

1  REM "PIE"
10  LET E=40
20  LET X=1
30  GOSUB Y000    [to Subroutine 9.11, PIE GRAPH PLOT]
40  FOR K=1 TO 8
50  LET S=E
60  LET E=S+40
70  LET X=X + 1
80  IF X=3 THEN GOTO 100
90  GOSUB Z000    [to Subroutine 9.12, PIE SEGMENT]
100 IF X=3 THEN LET X=0
110 NEXT K
120 STOP

```

Subroutine 9.13 **PRINTS** (instead of **PLOTS**) the pie graph. The character to be printed is input as X\$. This variable printed

character provides an almost infinite variety of distinctive segments. Either a character or a symbol which is associated with the segment can be used. For instance, when showing the distribution of income, you might print the mortgage payment segment in M's or **M**'s, the clothing segment in C's, auto with A's, savings with S's, and so on. This subroutine has many possibilities.

The graph subroutines in this chapter will provide you with a variety of possible output displays. With them, you can plot a histogram of class scores, or a point graph of statistical data. The PLOTTED POINT GRAPH (9.2) is an excellent display output for the TRENDS Subroutine (6.15). With manipulation of the print character in Subroutine 9.3, you can compare two or three sets of data on one graph. The possibilities with these subroutines are numerous.

### **PLOT With the T/S 2000 and Sinclair Spectrum**

The Timex/Sinclair 2000 and Sinclair Spectrum color computers provide a **PLOT** pixel which is one-quarter the size of the pixel used in the Timex/Sinclair 1000 or ZX81. The color computers also provide two very useful functions: **DRAW** and **CIRCLE**.

In order to use the **PLOT** subroutine in this book with the color computers, the dimensions must be changed. For example, in Subroutine 9.2, a maximum of 120 values can be plotted, rather than 30, so every **PLOT** number in the program must be multiplied by 4.

In general, you will want to use the **PLOT** subroutines only as guidelines, and design your own subroutines using the more sophisticated functions of the T/S 2000 and the Spectrum. For example, the **DRAW** function in the color computers replaces Subroutine 9.9, and most of Subroutine 9.8a can be replaced by use of the **CIRCLE** function. Similarly, Subroutine 9.10 becomes:

```
X00  REM PIE GRAPH CIRCLE
X10  CIRCLE 128, 88, 82
X20  RETURN
```



The Pie Graph in Subroutines 9.11 and 9.12 can become works of art with **DRAW**, **CIRCLE**, **OVER**, **INVERSE**, and so on.

You will certainly want to modify some of the **PRINT** graphing subroutines to take advantage of the color capabilities of these computers. These **PRINT** modifications are not necessary in order to use the **PRINT** subroutines, but will certainly provide you with practice with using your computer's facilities.

### Conclusion

Many of the subroutines in this chapter contain basic ideas or methods which, when used, will produce good results. If you take the time to understand these ideas, then as you use your own ingenuity to manipulate them, you will be able to evolve other output subroutines which will fit your needs more exactly.

## POINT GRAPH

**PURPOSE:** To Plot Points on a Cartesian Graph System

**9.1 NAME:** Printed Point Graph.

**INPUT VARIABLES:** A(I) = an array of values to be plotted (a maximum of 30 values can be plotted); MAX = the maximum value contained in A(I); N = the number of values in A(I).

**OUTPUT VARIABLES:** None.

```

X00  REM PRINT POINT
X01  REM INPUT A(I), MAX, N
X10  IF N>30 THEN LET N=30
X20  FOR I=1 TO 31
X30  IF I<=N THEN PRINT AT 19-A(I)*20/
      MAX,1+I*30/N;"."
X40  PRINT AT 20,I;"■"
X50  IF I<20 THEN PRINT AT I,1;"■"
```

```

X60 IF I <= 7 THEN PRINT AT 21,1+4*I; INT
    (4*I*N/30)
X70 NEXT I
X80 PRINT AT 0,0;MAX;" > "; AT 9,0;" > "; AT
    19,0;" 0"
X90 RETURN

```

## 9.2 NAME: Plotted Point Graph.

INPUT VARIABLES: A(I) = an array of values to be plotted (a maximum of 63 values can be plotted); MAX = the maximum value contained in A(I); N = the number of values in A(I).

OUTPUT VARIABLES: None.

```

X00 REM PLOT POINT
X01 REM INPUT A(I), MAX, N
X10 IF N > 63 THEN LET N=63
X20 FOR I=1 TO 63
X30 IF I < N THEN PLOT I*63/N,3+A(I)*40/MAX
X40 IF I < 42 THEN PLOT 0,I+2
X50 PLOT I,2
X60 NEXT I
X70 PRINT AT 0,0;MAX;" > "
X80 RETURN

```

## BAR GRAPH

**PURPOSE:** To Construct a Bar Graph, or Histogram

## 9.3 NAME: Printed Bar Graph.

INPUT VARIABLES: A(I) = an array of values to be plotted (a maximum of 30 values can be plotted); MAX = the maximum value contained in


A(I); N = the number of values in A(I); A = the number which will determine the character to be printed.

OUTPUT VARIABLES: None.

```

X00  REM PRINT BAR
X01  REM INPUT A(I), MAX, N, A
X10  IF N>30 THEN LET N=30
X20  FOR I=1 TO N
X30  FOR X=1 TO A(I)
X40  PRINT AT 20-INT (X*20/
      MAX+0.5),INT(I*30/N+0.5);CHR$(A)
X50  NEXT X
X60  NEXT I
X70  RETURN

```

NOTE: If desired, CHR\$(A) in line X40 can be replaced by a fixed character, such as . Alternately, each bar can be printed with a different character by adding:

```

X55  LET A=A+1

```

#### 9.4 NAME: Plotted Bar Graph.

INPUT VARIABLES: A(I) = an array of values to be plotted (a maximum of 60 values can be plotted); MAX = the maximum value contained in A(I); N = the number of values in A(I).

OUTPUT VARIABLES: None.

```

X00  REM PLOT BAR
X01  REM INPUT A(I), N, MAX
X10  IF N>60 THEN LET N=60
X20  FOR X=1 TO N
X30  FOR Y=1 TO A(X)*40/MAX+0.5
X40  PLOT INT(X*60/N+0.5),Y

```

```

X50  IF N<21 THEN PLOT INT(X*60/N+0.5)+1,Y
X60  NEXT Y
X70  NEXT X
X80  RETURN

```

NOTE: This subroutine will plot up to 60 bars. When less than 20 bars are being plotted, the bars are printed double width. If desired, scale can be added by including the following lines:

```

X75  PRINT AT 0,0; MAX;" > "

```

## HORIZONTAL BAR GRAPH

**PURPOSE:** To Construct a Horizontal Histogram

### 9.5 NAME: Horizontal Printed Bar Graph.

**INPUT VARIABLES:** A(I) = an array of values to be plotted (a maximum of 21 values can be plotted); MAX = the maximum value contained in A(I); N = the number of values in A(I).

**OUTPUT VARIABLES:** None.

```

X00  REM HORIZ BAR PRINT
X01  REM INPUT A(I),MAX,N
X10  IF N>21 THEN LET N=21
X20  FOR I=1 TO N
X30  FOR J=1 TO A(I)*30/MAX+0.5
X40  PRINT AT I*21/N-1,J-1;"█";
X50  NEXT J
X60  IF INT(A(I)*30/MAX+0.5)+LEN
    STR$A(I)<=32 THEN PRINT A(I)
X70  NEXT I
X80  PRINT AT 21,6;"MAX=";MAX
X90  RETURN

```

**9.6 NAME: Horizontal Plotted Bar Graph.**

**INPUT VARIABLES:**  $A(I)$  = an array of values to be plotted (a maximum of 44 values can be plotted);  $MAX$  = the maximum value contained in  $A(I)$ ;  $N$  = the number of values in  $A(I)$ .

**OUTPUT VARIABLES:** None.

```

X00  REM HORIZ BAR PLOT
X01  REM INPUT A(I),MAX,N
X10  IF N>44 THEN LET N=44
X20  FOR I=1 TO N
X30  FOR J=1 TO A(I)*40/MAX+0.5
X40  PLOT J,44-I*43/N
X50  IF N<15 THEN PLOT J,43-I*43/N
X60  NEXT J
X70  NEXT I
X80  RETURN

```

**COORDINATE SYSTEMS PLOT**

**PURPOSE:** To Plot Points on a Chosen Type of Graph

**9.7a NAME: Cartesian Coordinates (Four-Quadrant).**

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```

X00  REM CARTESIAN GRAPH
X10  FOR I=0 TO 31
X20  IF I<22 THEN PRINT AT I,15;"+"
X30  PRINT AT 10,I;"+"
X40  NEXT I
X50  RETURN

```

**9.7b** NAME: Plotting in Cartesian Coordinates (Four-Quadrant).

INPUT VARIABLES: X = horizontal coordinate;  
Y = vertical coordinate.

OUTPUT VARIABLES: None.

```
X00  REM CARTESIAN PLOT
X01  REM INPUT X AND Y
X10  IF X>32 OR X<-31 OR Y>21 OR
      Y<-22 THEN RETURN
X20  LET X=31+X
X30  LET Y=22+Y
X40  PLOT X,Y
X50  RETURN
```

**9.8a** NAME: Polar Coordinates.

INPUT VARIABLES: None.

OUTPUT VARIABLES: None.

```
X00  REM POLAR GRAPH
X10  LET P=PI/180
X15  FOR J=5 TO 15 STEP 5
X20  FOR I=P*10 TO P*360 STEP P*10
X30  IF J=15 AND (I>50*P AND I<130*P OR
      I>220*P AND I<320*P) THEN GOTO X50
X40  PRINT AT 10+J*SIN I,15+J*COS I;"+"
X50  NEXT I
X55  NEXT J
X60  FOR I=0 TO 31
X70  IF I<22 THEN PRINT AT I,15;"+"
X75  PRINT AT 10,I;"+"
X80  NEXT I
X90  RETURN
```

**9.8b** NAME: Plotting in Polar Coordinates.  
 INPUT VARIABLES: AN = angle up from right  
 horizontal; DS = distance out from origin.  
 OUTPUT VARIABLES: None.

```

X00  REM POLAR PLOT
X01  REM INPUT AN AND DS
X10  LET Y=22+DS*SIN(PI*AN/180)
X20  LET X=31+DS*COS(PI*AN/180)
X30  IF X>63 OR X<0 OR Y>43 OR Y<0
      THEN RETURN
X40  PLOT X,Y
X50  RETURN
  
```

## CARTESIAN LINE PLOT

**PURPOSE:** To Plot a Line between Two Cartesian Points

**9.9** NAME: From-To Plot.  
 INPUT VARIABLES: X1 = x coordinate of  
 starting point; Y1 = y coordinate of starting point;  
 X2 = x coordinate of ending point; Y2 = y  
 coordinate of ending point.  
 OUTPUT VARIABLES: None.

```

X00  REM FROM-TO
X01  REM INPUT X1,Y1,X2,Y2
X10  LET X=X2-X1
X15  LET Y=Y2-Y1
X20  IF X=0 THEN GOTO X70
X30  LET T= ATN(Y/X)
X35  LET X1=X1+(SGN X)/4
X40  IF X1<X2+0.1 AND X1>X2-0.1 THEN
      RETURN
  
```

```

X50  PLOT X1,(Y1+(X1-X2+X)*TAN T)
X60  GOTO X35
X70  LET Y1=Y1+SGN Y
X80  PLOT X1,Y1
X85  IF Y1=Y2 THEN RETURN
X90  GOTO X70

```

**NOTE:** This subroutine plots only positive values of X and Y. To change the coordinate system to a four quadrant one, change lines X50 and X80:

```

X50  PLOT 31+X1,22+(Y1+(X1-
      X2+X)*TAN T)
X80  PLOT 31+X1,22+Y1

```

## PIE GRAPH

**PURPOSE:** To Print a Pie Graph Using Symbols

**9.10 NAME:** Pie Graph Outline for Printed Graph.

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```

X00  REM PIE GRAPH CIRCLE
X01  REM **
X10  FOR J=0 TO 360 STEP 3
X20  PLOT (20.5*SIN(J/180*PI)+.5)+
      32,(20.5*COS(J/180*PI)+.5)+22
X30  NEXT J
X40  RETURN

```



**PURPOSE:** To Plot a "Pie" Graph

**9.11 NAME:** Two-Part Pie Graph.

**INPUT VARIABLES:** E = arc (in degrees) to be plotted.

**OUTPUT VARIABLES:** None.

```

X00  REM PIE GRAPH PLOT
X01  REM INPUT E
X10  LET V=360
X20  FOR I=20 TO 1 STEP -1
X30  FOR J=0 TO V STEP 3
X40  PLOT I*SIN(J/180*PI)+30,I*COS(J/180
      *PI)+20
X50  NEXT J
X60  LET V=E
X70  NEXT I
X80  RETURN

```

**9.12 NAME:** Pie Segment Plot.

**INPUT VARIABLES:** S = start of segment (in degrees); E = end of segment (in degrees); X = plot spacing (1 = solid, 2 = checked):

```

X00  REM PIE SEGMENT
X01  REM INPUT S,E, AND X
X10  FOR I=19 TO 1 STEP -X
X20  FOR J=S TO E STEP 3
X30  PLOT I*SIN(J/180*PI)+30,I*COS(J/180
      *PI)+20
X40  NEXT J
X50  NEXT I
X60  RETURN

```

**NOTE:** Subroutines 9.11 and 9.12 are designed to work together. Subroutine 9.11 draws a circle and plots one segment. Subroutine 9.12 plots additional segments, and can be used without 9.11 if no circle is needed.

**PURPOSE:** To Print a Pie Graph Using Symbols

**9.13 NAME:** Pie Graph Segment Print.

**INPUT VARIABLES:** S = start of segment (in degrees); E = end of segment (in degrees); X\$ = character or graphic to be used for plot.

**OUTPUT VARIABLES:** None.

```

X00  REM PIE PRINT
X01  REM INPUT S,E, AND X$
X10  FOR I=1 TO 10
X20  FOR J=S TO E STEP 5
X30  PRINT AT I*SIN(J/180*PI)+10,I*COS(J/180
      *PI)+16;X$
X40  NEXT J
X50  NEXT I
X60  RETURN

```

**NOTE:** X\$ can be any character you choose. Try graphic symbols, but also try letters and punctuation marks.

# Chapter 10

## Tables

The subroutines in this chapter are designed to display your data in a number of table-like formats. These tables make it easy to view a mass of data in an organized form. The displayed data may be interrelated or discontinuous. An example of interrelated data would be a table showing interest paid over a period of years for different interest rates. (The table rows would represent years, while each column would show a different rate.) An example of discontinuous data might be the comparison of the performance of different employees who work in different departments. The employees could be listed as rows, and the different departments could be listed as columns.

The data may be generated by your main program (as in the case of the interrelated example above) or may be input, point by point (as in the case of the discontinuous example). When the table is in your program memory, it then can be updated and modified, and can be used with other subroutines for other types of outputs.

If you are using a T/S 2000 or a Sinclair Spectrum computer, you may want to modify the subroutines to include identifying colors for headings, data categories, etc. It should also be pointed out that the ledger sheet subroutines which contain a **GOSUB** to Subroutine 8.7 (see Subroutines 10.2, 10.3, 10.4, and 10.5) must be modified if you are using a

color computer. This modification is the removal of the three lines related to **SCROLLing**:

```
LET R=24-PEEK (16442)
IF R=22 THEN GOSUB Y000
IF R=22 THEN GOTO X100
```

With these lines removed, you can **SCROLL** manually whenever **SCROLL** is needed, but the heading will not be saved.

### Ledger Sheets

The first five subroutines in this chapter relate to a special type of table called a *ledger sheet*. The ledger sheet allows you to identify the source of an entry by name and to enter a debit or credit value on the same line as the name. The ledger sheet can be used as a normal business ledger or as the basis for a checkbook register. Subroutine 10.1 prints the ledger sheet form, and should be used with all of the other ledger sheet subroutines.

Subroutines 10.2 and 10.3 use a two-dimensional string array for data storage. This meaning of “two-dimensional string array” is not difficult to grasp: The word “string” tells us that the array will accept both alphabetical and numerical data. “Array” says that the string is divided into specific segments and will be both limited and defined by a **DIM** statement in your main program. “Two-dimensional” means that the string can change in only two ways. This two-dimensional string can be illustrated by drawing a rectangle as shown in figure 10-1. The rectangle has two dimensions—height and width. Each of the 24 boxes can be uniquely identified by number. If we compare this rectangle to a string array, then each of the small boxes can represent the storage position of a single character in the array **A\$(4,6)**. The first dimension of **A\$(4,6)** has 4 elements, and the second dimension has six. The array would be formed by the statement **DIM A\$(4,6)**.

Each “element” of the 4 by 6 array can be individually addressed. You can print a single element such as **A\$(2,1)** or a row of elements such as **A\$(2, 1 TO 6)**.

Such a string array has many advantages. In the case of

		Column					
		1	2	3	4	5	6
Row	1	1,1	1,2	1,3	1,4	1,5	1,6
	2	2,1	2,2	2,3	2,4	2,5	2,6
	3	3,1	3,2	3,3	3,4	3,5	3,6
	4	4,1	4,2	4,3	4,4	4,5	4,6

**Figure 10-1** Representation of the array formed by the program statement **DIM A\$(4,6)**.

our example, we can store four sets of items, where each item has up to six characters, and we can address each character by number. If we say **PRINT A\$(2)**, the computer will print the six characters stored in the array positions of: 2,1, 2,2, 2,3, 2,4, 2,5, and 2,6.

The disadvantage of a string array is that you must dimension the array at the beginning of the program. This sets an upper limit on the size of the array. If you are using our example rectangle array, **A\$(4,6)**, to list the names of your friends, you have dimensioned the array with a statement, **DIM A\$(4,6)**. Then you would enter your friends names:

ALICE  
ROBERT  
JANE  
TARZAN

Because the second dimension of the array is 6, any name longer than six characters would have to be abbreviated. (A name like Shirley, for example, would need to be input as Shirly). Also, this list of names is fine as long as you have only four friends. As soon as you make a fifth friend, however, you will find that the array will not hold the new friend's name, because the array's first dimension is 4. If you had considered this limitation earlier, and had dimensioned your array with **DIM A\$(100,6)**, then it would have been no problem to add a name to the list. However, with this larger array, you have tied up 600 spaces in your computer's memory and,

so far, with this larger array have used only 30 of the spaces for five names.

The format for using a two-dimensional string array with the ledger sheet is shown in Subroutine 10.2. The first dimension is the data number and the second dimension is the twenty-five data characters. This format requires that you dimension your array in the following form:

```
DIM L$(N,25)
```

In this example, N is the maximum number of ledger sheet entries you will need (like the maximum number of friends in the earlier example).

Subroutine 10.2 will print on the ledger form the data which is stored in L\$(N,25). This subroutine will print one line at a time and if the bottom line is filled, it then will jump automatically, to a scroll subroutine, before printing. If you are using the color computers, this automatic **SCROLL** function cannot be used, and lines X10, X20, and X30 should be deleted. (With these lines deleted, you can **SCROLL** manually.)

Subroutine 10.3 will accept a new entry into the ledger data. There are few prompts, so you must remember what input is required. The first prompt is the new line number, printed on the ledger sheet by subroutine line X25. The appearance of this line number on the ledger means that you can input up to thirteen characters for the entry name. When this input is printed on the ledger sheet, you then must input either a 0 for a debit entry code or a 1 for a credit entry code. There is no prompt for this entry and the subroutine will accept only a 0 or 1 as input (see line X55). Now, the program expects you to input (without a prompt) the value of your entry. This value can be no more than seven characters long, including the decimal point, if any. These seven characters allow you to enter up to 9999.99. Notice, if you want to keep the decimal points in line, then initial spaces must be entered on smaller numbers. Thus:

```
" _126.22"
```

```
"10001.00"
```

```
" _ _ _5.21"
```

Entering 5.21 without spaces will print as 5.21 \_ \_ \_ , since the array will fill in the spaces to the right of the entry.

You might want to include in your program the following subroutine controls:

```

      .
      .
      .
210  GOSUB      [to Subroutine 10.1 and print the form]
220  FOR N=1 TO 20
230  IF L$(N,1)=" _ " THEN GOTO 260
240  GOSUB      [to Subroutine 10.2 and Print Line "N"]
250  GOTO 270
260  GOSUB      [to Subroutine 10.3 and input next line]
270  NEXT N
      .
      .
      .

```

The program segment above will give you one idea of how to link the first three subroutines together. Such a program segment assumes that L\$ has been dimensioned as L\$(20,25), and may contain some previously stored lines.

Subroutines 10.4 and 10.5 are the storage string counterparts of Subroutines 10.2 and 10.3. Subroutine 10.4 prints stored data on the ledger sheet form, and Subroutine 10.5 accepts new data to be printed and stored. These subroutines work exactly the way 10.2 and 10.3 work, except that Subroutines 10.4 and 10.5 store the data in information storage strings (see Chapter 6), rather than in string arrays.

### Matrix Tables

The remaining subroutines in this chapter (10.6 through 10.15) all relate to tables which display 4 columns of data, with each column being 19 or 20 lines deep. The table element defined by a column/line can have up to 6 characters of data in it. Depending upon the data storage method which you use, the

table data can be either only numerical or both alphabetical and numerical.

The last table in this chapter is called "Super-Matrix." It is designed for numerical data only, but allows you to construct a 12-column by 40-line table and to view any 4-column by 19-line portion of this "super" table.

The regular matrix tables, covered in Subroutines 10.6 to 10.11, will allow you to use any one of three data storage methods. These methods are the array, the string array, and the storage string. You should choose the data storage method which best suits your needs and then use the subroutines which match your chosen storage method.

Subroutine 10.6 (PRINT MATRIX TABLE) will print the table of a two-dimensional numerical string. This string is dimensioned as A(4,20), with the first dimension identifying the column and the second dimension identifying the line. Subroutine 10.7 (MATRIX TABLE INPUT) is designed to work with Subroutine 10.6. Subroutine 10.7 allows you to input new data to one element of the table or to change an existing table element. No more than 6 characters should be printed in a table element. (This limit includes the space needed for any decimal point which may be required.)

Subroutines 10.8 and 10.9 are a pair of subroutines which will print and input table information in an information storage string (see Chapter 6). The PRINT STRING TABLE (10.8) prints 20 lines of 25 character substrings. Each substring contains the information for 6 characters in each of 4 columns on the row described by that substring. The input subroutine (10.9) allows you to input or change data in a 6-character block of a substring.

Subroutines 10.10 and 10.11 are for use with a string array method of data storage. The string array is three-dimensional. The first dimension is the column number, the second is the row number, and the third dimension is the 6 character elements to be printed in the row/column "box." Subroutine 10.10 prints the data table, and Subroutine 10.11 provides for input or change of a table data element.

The Super-Matrix is a table 12 columns wide and 40 lines high. The computer cannot print the entire table on the screen, so Subroutine 10.12 (WINDOW) prints a 4-column by 19-row



segment of, or window on, the table. Thus, you could view rows 9 through 27 of columns 5, 6, 7, and 8. Or, perhaps you are interested in the data shown in rows 16 through 34 of columns 8, 9, 10, and 11. The data to be printed is held in a 12- by 40-element, two-dimensional numerical array, A(12,40).

Printing the "window" portion of the table in Subroutine 10.12 is perfectly adequate, but in order to *move* the window over one column or down one row, you must reprint the entire screen. If you are searching the table for a specific piece of data, reprinting the entire window may be very time consuming. We can, of course, **SCROLL** in order to move the window down one row, but there is no Timex/Sinclair BASIC function for scrolling "sideways" to the next column. Since we need to scroll sideways across the screen, we must construct such a scrolling subroutine.

First, we need a "key" which will cause the new function to operate when called. When the ENTER key is pressed, Subroutine 10.13 (C-SHIFT KEY) will shift columns 2, 3, and 4 into positions 1, 2, and 3, and then will print new data in column 4. This operation effectively shifts the window to the left on the table. The KEY subroutine does not actually do the shifting. It merely identifies the pressing of the ENTER key, goes to another subroutine to shift the columns, and then does the clean-up job of printing the new column 4 and of changing the column numbers along the heading.

Subroutine 10.14 (C/R-SHIFT KEY) is an embellishment of the C-SHIFT KEY subroutine. C/R KEY will accept an input of C or R. If C is input, this subroutine scrolls sideways, exactly as does the C-KEY subroutine. If R is input, then the C/R-KEY goes to a **SCROLL** subroutine to reveal another row of data. Thus, by using Subroutine 10.14 you can alternately C-SCROLL and R-SCROLL your way from left to right and from top to bottom of the "Super-Matrix Table."

Line X70 of Subroutine 10.14 will not work with the color computers. With these computers, it will probably be necessary to **POKE** 23692,0 to hold the table window still, and **POKE** 23692,1 in order to get the opportunity to manually **SCROLL** the table.

Subroutine 10.15 will not work with the color computers,

either, because it uses machine code designed for the noncolor computers. We suggest that if you have a color computer you should avoid the roll and scroll subroutines (10.13, 10.14, and 10.15), and use only the WINDOW subroutine (10.12). By inputting the column and row wanted, the window can be manipulated to any desired position on the table, and will work quite well with the color computers.

The SHIFT TABLE LEFT subroutine (10.15) is the only subroutine in this book which uses a machine code program. We hope that this subroutine will encourage you to learn more about the use of machine code in constructing your programs.

The machine code program uses a series of numbers which the computer recognizes as machine commands. The function **USR** (see line X80) tells the computer to execute the machine code found in the memory location which follows the function **USR**. In line X80 that memory location is  $V+3$ . The rest of the subroutine determines the value of  $V$ .

The machine code program is stored in the string  $A\$$  (see line X20). When the computer sees numbers and has not been told to **USR**, it prints the number as characters. That is why line X20 appears to be so strange. The first code number is 62, which the computer prints as Y. The next number is 19, printed as <. The next is 42, an E, and so on. The table for the conversion of numbers to print characters is found in your computer manual under the title "The Character Set".

The page following Subroutine 10.15 is the step-by-step method for entering the characters of machine code into the string  $A\$$ . Follow the steps carefully, and you will have no trouble. The page which follows the step-by-step input lists the sequence (1 through 31) of the code steps and also lists the Assembler Language, which is often used to describe machine code programs.

After the code is entered, we must be able to tell the **USR** function where the code is found. Line X10 sets  $V$  equal to the memory location of the variables file. The code number for  $A\$$  in the variable file is 70. So, we are now looking through the file for variable 70. If we find it (see line X25), then we can immediately jump to line X80. Otherwise, we must plod through all of the other variables until we find it

(this plodding would include lines X30, X40, X50, X60, and X70).

When the variable is found and the code is called by the **USR** function, the code looks into the display file (to determine what is on the tube), shifts the last 19 rows over 9 spaces (1 column's width), and returns to the subroutine. The subroutine requires a few seconds to find A\$, but when A\$ is found, the actual machine code does its work in milliseconds.

### Caution

Because the machine code directly manipulates computer memory, a mistake in typing the code can cause your system to "crash," and you will lose any program in the system at the time. It is suggested that after entering Subroutine 10.15 into your program, you save your entire program on tape *before* testing the subroutine. By saving the program in this manner, you avoid losing the many hours of typing which you have already invested in your program.

### Conclusion

The subroutines in this book are designed to pique your curiosity and to provide you with useful information for program construction. As we have implied throughout this book, the purpose of the book is to help you to learn to write your own subroutines.

All of the methods used here are described in your computer manual. The problem is that to be completely understood, some of the descriptions require more study. The first step toward understanding these methods is to understand how your computer works and then to experiment with it. There are many books available on the basics of computer operation. There are also a few "basic" books printed on the Z-80 microcomputer chip, which is the core of the Timex/Sinclair Computer.

The Timex/Sinclair Computer is an extremely versatile machine. How versatile it can become for you will depend upon your programming skill and your understanding of the

machine. We hope this book will help you to build better, more useful programs.

## LEDGER SHEET

**PURPOSE:** To Print a Ledger Sheet Form for Use with Subroutines 10.2, 10.3, 10.4, and 10.5

**10.1 NAME:** Ledger Sheet Form.  
**INPUT VARIABLES:** None.  
**OUTPUT VARIABLES:** None.

```
X00 REM LEDGER FORM
X10 PRINT AT 0,6;"ENTRY"; TAB 18;"DEBIT";
    TAB 26;"CREDIT"
X20 FOR I=0 TO 21
X30 PRINT AT I,2;":"; TAB 16;":"; TAB 24;":"
X40 PRINT AT I,1;"-"; TAB I+10;"-"
X50 NEXT I
X60 RETURN
```

**PURPOSE:** To Print Information Stored in a String Array onto a Ledger Form

**10.2 NAME:** Print String Array on Ledger Sheet.  
**INPUT VARIABLES:** L\$(I,23) = a two-dimensional string array with the first dimension the data number and the second dimension the data; N = the data number to be printed.  
**OUTPUT VARIABLES:** None.

```
X00 REM ARRAY LEDGER PRINT
X01 REM INPUT L$(I,23) AND N
X10 LET R=24-PEEK (16442)
X20 IF R=22 THEN GOSUB Y00
X30 IF R=22 THEN GOTO X10
```

[to Subroutine  
 8.7, SCROLL  
 with heading]

```

X40  PRINT AT R,0;L$(N,1 TO 2);
X50  PRINT TAB 2;";";L$(N,3 TO 15);
X60  PRINT TAB 16;";"; TAB 17+VAL
      L$(N,16)*8;L$(N,17 TO 23)
X70  PRINT AT R,24;";"
X90  RETURN

```

NOTE 1: The data format for L\$ is:

Place:	1, 2,	3 . . . 15,	16,	17 . . . 23
	<u>          </u>	<u>          </u>	<u>          </u>	<u>          </u>
Use:	Item	Item	D/C	Item
	number	name	code	value

NOTE 2: The debit credit code for place 16 is:

0 = Debit entry

1 = Credit entry

**PURPOSE:** To Input Data into a String Array and  
Print it on a Ledger Sheet

**10.3 NAME:** Input and Print String Array on Ledger  
Sheet.

**INPUT VARIABLES:** L\$(I,23) = a two-  
dimensional string array with the first dimension  
the data number and the second dimension the  
data; N = the data number to be input and  
printed.

**OUTPUT VARIABLES:** L\$(I,23) = the data array  
with the added input.

```

X00  REM ARRAY LEDGER INPUT
X01  REM INPUT L$(I,23) AND N
X02  REM OUTPUT L$(I,23)
X10  LET R=24-PEEK (16442)

```

```

X15  IF R=22 THEN GOSUB Y000 [to Subroutine
                                8.7, SCROLL
                                with heading]
X20  IF R=22 THEN GOTO X10
X25  PRINT AT R,0;N;
X30  LET L$(N,1 TO 2)= STR$ N
X35  INPUT L$(N, 3 TO 15)
X40  PRINT TAB 2;"":L$(N,3 TO 15); TAB
      16;"":;
X50  INPUT L$(N,16)
X55  IF L$(N,16)<>"0" AND L$(N,16)<>"1"
      THEN GOTO X50
X60  INPUT L$(N,17 TO 23)
X70  PRINT TAB 17+VAL L$(N,16)*8;L$(N,17
      TO 23); AT R,24;"":
X90  RETURN

```

NOTE: See notes on Subroutine 10.2.

**PURPOSE:** To Print Information Stored in an Information String onto a Ledger Sheet

- 10.4 NAME:** Print Information String on Ledger Sheet.  
**INPUT VARIABLES:** I\$ = information string with subgroup length C equal to 25; N = subgroup number to be printed.  
**OUTPUT VARIABLES:** None.

```

X00  REM STRING LEDGER PRINT
X01  REM INPUT I$ AND N
X10  LET R=24-PEEK (16442)
X20  IF R=22 THEN GOSUB Y000 [to Subroutine
                                8.7, SCROLL
                                with heading]
X30  IF R=22 THEN GOTO X10
X40  PRINT AT R,0;I$(25*N-24 TO 25*N-23);
X50  PRINT TAB 2;"":I$(25*N-22 TO
      25*N-10);

```

X60 **PRINT TAB 16;" "; TAB 17+VAL**  
 $I\$(25*N-9)*8;I\$(25*N-8 \text{ TO } 25*N-2)$   
 X70 **PRINT AT R,24;" "**  
 X90 **RETURN**

NOTE 1: The substring format for I\$ is:

Place: 1, 2,    3 . . . 15,    16,    17 . . . 23    24, 25

Use:	Item	Item	D/C	Item	Spaces
	number	name	code	value	

NOTE 2: The debit/credit code for place 16 is:

0 = Debit entry

1 = Credit entry

**PURPOSE:** To Input Data into an Information String and Print it on a Ledger Sheet

### 10.5 NAME: Input and Print Information String on Ledger Sheet.

INPUT VARIABLES: I\$ = information string with subgroup length C equal to 25; N = number of subgroup to be input and printed.

OUTPUT VARIABLES: I\$ = information string including new input.

X00 **REM STRING LEDGER INPUT**

X01 **REM INPUT I\$ AND N**

X02 **REM OUTPUT I\$**

X10 **LET R=24-PEEK (16442)**

X15 **IF R=22 THEN GOSUB Y00**

[to Subroutine  
8.7, SCROLL  
with heading]

X20 **IF R=22 THEN GOTO X10**

X25 **DIM X\$(25)**

X30 **PRINT AT R,0;N;**

X35 **LET X\$(1 TO 2)= STR\$ N**

```

X40 INPUT X$(3 TO 15)
X45 PRINT TAB 2;" ":"X$(3 TO 15); TAB 16;" ":";
X50 INPUT X$(16)
X55 IF X$(16) < > "1" AND X$(16) < > "0" THEN
      GOTO X50
X60 INPUT X$(17 TO 23)
X70 PRINT TAB 17+VAL X$(16)*8;X$(17 TO
      23); AT R,24;" ":"
X80 LET I$=I$+X$
X90 RETURN

```

## MATRIX TABLE

**PURPOSE:** To Construct a 4-Column, 20-Line Table Using an Array

**10.6 NAME:** Printing a Table with Data Storage in an Array.

**INPUT VARIABLES:** A(4,20) = a data array containing four elements in the first dimension and 20 elements in the second.

**OUTPUT VARIABLES:** None.

```

X00 REM PRINT MATRIX TABLE
X01 REM INPUT A(4,20)
X10 PRINT "NO.: _COL-1: _COL-2: _COL-3: _
      COL-4:"
X20 PRINT " : : [32 colons] : : "
X30 FOR J=1 TO 20
X40 PRINT AT J+1,0;J; TAB 3;" ":";
X50 FOR I=1 TO 4
X60 PRINT A(I,J); TAB I*7+3;" ":";
X70 NEXT I

```



X80 **NEXT J**

X90 **RETURN**

NOTE 1: This subroutine assumes that A(4,20) already exists due to the program statement **DIM A(4,20)**. The array A(4,20) may contain data generated by some function, or may contain discontinuous data inserted by Subroutine 10.7.

NOTE 2: If the right-margin justification in columns is desired, then change line X60 to read:

X60 **PRINT TAB I\*7+3-LEN STR\$ A(I,J);A(I,J);**  
**TAB I\*7+3;" ";**

**10.7 NAME:** Input to Change One Element of a 4 by 20 Matrix.

**INPUT VARIABLES:** A(4,20) = a data array containing a 4 by 20 matrix of numbers; C = column number of element to be changed; R = row number of element to be changed.

**OUTPUT VARIABLES:** A(4,20) = data array with a new element at A(C,L).

X00 **REM MATRIX TABLE INPUT**

X01 **REM INPUT A(4,20), C, R**

X02 **REM OUTPUT A(4,20)**

X10 **INPUT A(C,R)**

X20 **PRINT AT R+1,C\*7-3;" \_ \_ \_ \_ \_ "**

X30 **PRINT AT R+1,C\*7-3;A(C,R)**

X40 **RETURN**

NOTE 1: This subroutine may be used either to insert information into A(4,20), or to change existing information.

NOTE 2: To print right margin justified, change line X30 to read:

```
X30  PRINT AT R+1,C*7-3-LEN STR$
      A(C,R);A(C,R)
```

NOTE 3: If A(C,R) may be longer than six characters, you may wish to add the following:

```
X15  IF LEN STR$ A(C,R)>6 THEN LET VAL
      (STR$ A(C,R)(1 TO 6))
```

## STRING TABLE

**PURPOSE:** To Construct a 4-Column by 20-Line Table Using an Information String



**10.8 NAME:** Table Printing with an Information String Data Storage.

**INPUT VARIABLES:** I\$(500) = an information string containing 20 subsets of 25 characters each (see NOTE 1).

**OUTPUT VARIABLES:** None.

```
X000  REM PRINT STRING TABLE
X001  REM INPUT I$(500)
X100  PRINT "NO.: _COL-1: _COL-2: _COL-3: _
      COL-4:"
X200  PRINT " : : [32 colons]: : "
X300  FOR J=1 TO 20
X400  PRINT AT J+1,0;J; TAB 3;" : ";
X500  FOR I=1 TO 4
X600  PRINT I$(J*25+I*6-30 TO J*25+I*6-25);
      TAB I*7+3;" : ";
X700  NEXT I
X800  NEXT J
X900  RETURN
```

NOTE 1: The subsets of I\$ are arranged as follows:

Place:	1 . . . 6,	7 . . . 12,	13 . . . 18,	19 . . . 24,	25
					
Data:	Col-1	Col-2	Col-3	Col-4	Space

NOTE 2: Notice that both words and numbers can be inserted into the six character "boxes." Column 1 can be used for row names and row 1 can be used for column names, if this would be useful.

### 10.9 NAME: Input to Change a Six-Character Group within an Information String.

INPUT VARIABLES: I\$(500) = an information string containing 20 subsets of 25 characters each;  
C = column of table to be changed; R = row of table to be changed.

OUTPUT VARIABLES: I\$(500) = the information string with new data.

```

X00  REM STRING TABLE INPUT
X01  REM INPUT I$(500), C, R
X02  REM OUTPUT I$(500)
X10  DIM X$(6)
X20  INPUT X$
X30  LET I$(R*25+C*6-30 TO
      R*25+C*6-25)=X$
X40  PRINT AT R+1,C*7-3;I$(R*25+C*6-30
      TO R*25+C*6-25)
X50  RETURN

```

NOTE: This subroutine assumes that the string I\$(500) exists prior to the execution of the subroutine. This subroutine can be used to either input initial data into the string or to change existing data held in the string.

## STRING MATRIX TABLE

**PURPOSE:** To Construct a 4-Column by 20-Line Table Using a String Array

**10.10 NAME:** Table Printing with a String Array Data Storage.

**INPUT VARIABLES:** A\$(4,20,6) = a 4- by 20- by 6-character data string.

**OUTPUT VARIABLES:** None.

```

X00  REM PRINT STRING MATRIX
X01  REM INPUT A$(4,20,6)
X10  PRINT "NO.: _COL-1: _COL-2: _COL-3: _
      COL-4:"
X20  PRINT ": : :[32 colons]: : ."
X30  FOR J=1 TO 20
X40  PRINT AT J+1,0;J; TAB 3;":.";
X50  FOR I=1 TO 4
X60  PRINT TAB I*7-3;A$(I,J); TAB I*7+3;":.";
X70  NEXT I
X80  NEXT J
X90  RETURN

```

**10.11 NAME:** Input to Change Six Elements of an Information String Subset.

**INPUT VARIABLES:** A\$(4,20,6) = a 4- by 20- by 6-character data string; C = column of table to be changed; R = row to be changed.

**OUTPUT VARIABLES:** A\$(4,20,6) = the data string including the changed subset A\$(C,R,6).

```

X00  REM STRING MATRIX INPUT
X01  REM INPUT A$(4,20,6), C, R
X02  REM OUTPUT A$(4,20,6)

```

```

X10  PRINT AT R+1,C*7-3;" _ _ _ _ _ "
X20  INPUT A$(C,R,1 TO 6)
X30  PRINT AT R+1,C*7-3;A$(C,R)
X40  RETURN

```

## SUPER-MATRIX TABLE

**PURPOSE:** To Print A 4-Column by 19-Line "Window" on a 12-column by 40-Line Table

- 10.12 NAME:** Window on the Super-Matrix Table.  
**INPUT VARIABLES:** A(12,40) = an array (matrix) of values; C = the minimum column number to be printed; R = the minimum row number to be printed.

**OUTPUT VARIABLES:** None.

```

X00  REM WINDOW
X01  REM INPUT A(12,40), C, R
X10  PRINT AT 0,0;"COLUMNS_";C;" TO ";
      C+3;TAB 15;"__ROWS_";R;"
      TO ";R+18
X20  PRINT "ROW_:C-";C; TAB
      11;"C-";C+1; TAB 18;"C-";C+2; TAB
      25;"C-";C+3
X30  PRINT TAB 4;" + - - - [1 "plus" and 27
      "minus" signs] - - - "
X40  FOR I=0 TO 18
X50  PRINT AT I+3,2-LEN STR$(R+I);R+I
X60  FOR J=0 TO 3
X70  PRINT AT I+3,4+J*7;" : ";A(J+C,I+R)
X75  NEXT J
X80  NEXT I
X90  RETURN

```

**PURPOSE:** To Manipulate a 4-Column by 19-Line  
 "Window" on a 12-Column by 40-Line Table

**10.13 NAME:** Key for Shifting Columns on the Window.

**INPUT VARIABLES:** A(12,40) = an array  
 (matrix) of values; C = the minimum column  
 number of printed table; R = the minimum row  
 number of the printed table.

**OUTPUT VARIABLES:** C = new minimum  
 column number.

```

X00  REM C-SHIFT KEY
X01  REM INPUT A(12,40), C, R
X02  REM OUTPUT C
X10  INPUT Z$
X20  IF Z$ < > "" THEN RETURN
X30  GOSUB Y00      [to Subroutine 10.15, SHIFT TABLE
                    LEFT]
X40  LET C=C+1
X50  FOR I=0 TO 18
X60  PRINT AT I+3,25;" ";A(C+3,R+I)
X65  NEXT I
X70  PRINT AT 0,8;C;" TO ";C+3
X80  PRINT AT 1,7;C;AT 1,14;C+1;AT
      1,21;C+2;AT 1,28;C+3
X90  GOTO X10
  
```

**NOTE:** After Subroutine 10.12 has printed the win-  
 dow, this subroutine and Subroutine 10.15  
 will shift the window to the left when EN-  
 TER is pressed.

**10.14 NAME:** Key for Shifting Columns or Scrolling  
 Rows on the Window.

**INPUT VARIABLES:** A(12,40) = an array  
 (matrix) of values; C = the minimum column

number of the printed table; R = the minimum row number of the printed table.

OUTPUT VARIABLES: C or R = the new value of either C or R.

```


X00  REM C/R-SHIFT KEY
X01  REM INPUT A(12,40), C, R
X02  REM OUTPUT C OR R
X10  INPUT Z$
X15  IF Z$ <> "R" AND Z$ <> "C" THEN
      RETURN
X20  IF Z$ = "R" THEN GOTO X65
X30  GOSUB Y00 [to Subroutine 10.15, SHIFT TABLE
               LEFT]
X35  LET C=C+1
X40  FOR I=0 TO 18
X45  PRINT AT I+3,25;"":A(C+3,R+I)
X50  NEXT I
X55  PRINT AT 0,8;C;" TO ";C+3;AT 1,7;C;
      TAB 14;C+1; TAB 21;C+2; TAB 28;C+3
X60  GOTO X10
X65  LET R=R+1
X70  GOSUB Z00 [to Subroutine 8.7, SCROLL WITH
               HEADING]
X75  PRINT AT 0,22;R;" TO ";R+18;AT
      21,0;R+18
X80  PRINT AT 21,4;"":A(C,R+18); TAB
      11;"":A(C+1,R+18); TAB
      18;"":A(C+2,R+18); TAB 31;" ";AT
      21,25;"":A(C+3,R+18)
X90  GOTO X10

```

NOTE 1: Subroutine 10.14 accepts either an R or a C as input for Z\$. If Z\$ = "R", then the table is scrolled up one line. If Z\$ = "C",

then the table is shifted left one column.  
You may wish to add the following line as  
a reminder:

```
X57  PRINT AT 0,0;"INPUT _R TO SCROLL,  
      _C TO SHIFT"
```







NOTE 2: The Graphic symbol near the end of line X80 (TAB 31;""') is necessary in order to correct a problem which could be caused by the Timex/Sinclair **SCROLL** function. The symbol can be any printed character, but must be printed in position 31 of line 21, following a **SCROLL**. Without this printed position, the **SHIFT TABLE LEFT** subroutine (10.15) will not work properly when preceded by the **SCROLL** subroutine (8.7).

**PURPOSE:** To Shift Columns 2, 3, and 4 of the  
"Matrix Table Window" into Positions 1, 2,  
and 3

**10.15 NAME:** Shift Table Columns to the Left.

**INPUT VARIABLES:** None.

**OUTPUT VARIABLES:** None.

```
X00  REM SHIFT TABLE LEFT
X10  LET V=PEEK 16400+256*PEEK 16401
X20  LET A$="Y<E£RNDO_FAST
      SGN_=_GOSUBK$7<
      (UNPLOT X 4 PAUSE TAN"
X25  IF PEEK V=70 THEN GOTO X80
X30  IF (PEEK V>69 AND PEEK V<96) OR
      (PEEK V>133 AND PEEK V<160) OR
      (PEEK V>197 AND PEEK V<224) THEN
      LET V=V+3+PEEK(V+1)+256*PEEK(V+2)
```



```










X35  IF PEEK V>101 AND PEEK V<128 THEN
      LET V=V+6
X40  IF PEEK V>229 AND PEEK V<256 THEN
      LET V=V+18
X45  IF NOT (PEEK V>165 AND PEEK V<192)
      THEN GOTO X25
X50  LET V=V+1
X55  IF PEEK V>165 AND PEEK V<192 THEN
      GOTO X65
X60  GOTO X50
X65  LET V=V+6
X70  GOTO X25
X80  RAND USR (V+3)
X90  RETURN

```

NOTE 1: The strange-looking string of symbols appearing in line X20 [in the SHIFT TABLE LEFT subroutine (10.15)] are the characters and keywords representing the machine code which shifts the table. In order to work properly, the string must be input exactly as shown. Because some of the characters are normal, some are graphics, some are functions, and some are keywords, care must be taken to enter each character using the proper mode. The following chart will help you to enter each character in the correct mode and in the proper order.

Begin by entering the line number (X20), and the initial statement (**LET A\$=""**). Now, input the machine code by entering the next 31 steps in sequence. Finish the line with a quote mark (") and ENTER.

When you have finished entering the line X20, check it carefully against the line shown in Subroutine 10.15. Any errors can be corrected in the normal manner.

Step	Enter	Comment
1	Y	
2	<	
3	E	
4	£	
5	<b>RND</b>	Function mode, T
6		Graphics mode, shifted-1 (don't forget to leave graphic mode before continuing)
7	O	
8	(space)	Space
9		Graphics mode, shifted-D (don't leave graphic mode)
10		Graphics mode, shifted-D (now, leave graphic mode)
11	<b>FAST</b>	Letter mode, shifted-F
12	<b>SGN</b>	Function mode, F
13		Graphics mode, shifted-1
14		Graphics mode, shifted-E
15	(space)	Space
16		Graphics mode, shifted-D
17		Graphics mode, shifted-1
18	=	
19	(space)	Space
20	<b>GOSUB</b>	Keyword, see note below
21		Graphics mode, K
22		Graphics mode, shifted-T
23	\$	
24	7	
25	<	
26	(	
27	<b>UNPLOT</b>	Keyword, see note below
28	X	
29	4	
30	<b>PAUSE</b>	Keyword, see note below
31	<b>TAN</b>	Function mode, E

In order to enter a keyword, first enter **THEN** (shifted-3), enter the desired keyword, backspace (shifted-5), delete the **THEN** (shifted-0), and fore-space (shifted-8).






NOTE 2: If you are interested in the machine code in the preceding subroutine, the following table lists the characters, the code, and the Assembly language equivalent of the code.

Assembly language is a mnemonic code which represents the actions of the machine code on the registers. For instance, the machine code 62 tells the computer to load into register A, the number which follows. In Assembly language this action is expressed as **Ld A,N**.

Although most machine codes are expressed in a binary or a hexadecimal number base, the Timex/Sinclair computers use a decimal base code for input to the machine.

#### Machine Code for Subroutine 10.15, Shift Table Left

Step	Character	Decimal Code	Assembly Language Code
1	Y	62	Ld A,N
2	<	19	
3	E	42	Ld HL, (NN)
4	£	12	
5	RND	64	
6	■	1	Ld BC,NN
7	O	52	
8	(space)	0	
9	■	9	Add HL,BC
10	■	9	Add HL,BC
11	FAST	229	Push HL
12	SGN	209	Pop DE
13	■	1	Ld BC,NN

14		7	
15	(space)	0	
16		9	Add HL,BC
17		1	Ld BC,NN
18	=	20	
19	(space)	0	
20	<b>GOSUB</b>	237	} LDIR
21		176	
22		6	LD B,N
23	\$	13	
24	7	35	Inc HL
25	<	19	Inc DE
26	(	16	DJNZ
27	<b>UNPLOT</b>	252	(DIS)
28	X	61	Dec A
29	4	32	Jr NZ
30	<b>PAUSE</b>	242	(DIS)
31	<b>TAN</b>	201	Ret

---

# Index



# Index

- Acres, conversion:
  - to square kilometers, 43
  - to square yards, 45
- Alphabetizing, of information
  - substrings, 96
- Amortization, calculation of, 74-75
- Area:
  - circle, 24
  - ellipse, 24-25
  - polygon, 23
  - rectangle, 22-23
  - solids (surface area), 25-26
  - square, 21-22
  - triangle, 20-21
- Arrays:
  - inputting into, 51-54
  - median and mode of, 55-56, 58-59
  - minimum, maximum and mean of, 85, 89-90
- AUTO-SCROLL subroutine, 52, 103-104, 108-109
- Bar graph (histogram), 118-121
  - plotted, 119-120
  - printed, 118-119
- Bar graph, horizontal, 114
  - plotted, 121
  - printed, 120
- Base conversion, 47-50
- BASIC programs:
  - how to **SAVE**, 4-5
  - placement of subroutines in, 16-17
  - on Sinclair computers, 1
  - suggestions for keyboarding, 4-5
  - typographic conventions used in this book, 2-3
  - (*See also* Programming)
- Binary, conversion to decimal, 47
- Binary search, 97-98
  - vs. sequential search, 88
- Bubble sort, 86, 93
- Bubble sort and save, 86, 93-94
- Business, subroutines for, 66-83
- Cartesian coordinates:
  - displaying, 121-122
  - plotting within, 117-118, 121-124

- Celsius, conversion to Fahrenheit, 45
- Centimeters, conversion to inches, 34
- Central tendency (mean, median, mode) of an array, 58-59
- Characters:
  - in graphing, 151-153
  - inverse, 3
- Circle, area of, 24
- CIRCLE** function, for plotting, 116-117
- Color computers, Sinclair:
  - graphing on, 116-117
  - modifying subroutines for, 3-4, 104, 127-128
  - window subroutines on, 133-134
- Compound interest, maturity value at, 73-74
- Compound interest rate, if maturity known, 74
- Cone:
  - surface area of, 26
  - volume of, 28
- Conversions, 29-50
  - multiple, in one subroutine, 30
- Coordinates:
  - Cartesian, 117-118, 121-124
  - plotting within, 121-123
  - polar, 122-123
- Cube, volume of, 26
- Cursor **INKEY\$** subroutines, 100-102, 106
- Cylinder:
  - surface area, 25
  - volume, 27
- D-File system variable, 105
- Data:
  - discontinuous vs. interrelated, 127-128
  - generation of, by program, 127-128
  - input into an array, 51-56
  - input into a program, 127-128
  - SAVE** of, 4-5
  - storage string used for, 80-84
- Decimal conversion:
  - to binary, 47
  - to hex, 48, 49
  - of large numbers, 49
  - to other bases, 50
  - of small numbers, 47, 48
- Degrees, conversion to radians and grads, 46
- Depreciation methods:
  - applications of, 66-67
  - calculations, 77-79
- DIM** statement, for string array, 128-130
- Directory routine, 69, 70
- Discontinuous data, vs. interrelated data, 127-128
- Discount of payable notes:
  - bank, 72
  - true, 73
- Double-declining depreciation method, 78-79
- DRAW** function, for plotting, 116-117
- Efficiency, 8-9
  - (see also Memory, conserving)
- Ellipse, area of, 24-25
- English measures, converted to metric, 36
- Equilateral triangle, area of, 20
- F** mode, and trigonometric functions, 18



- Fahrenheit, conversion to Celsius, 45
- Fast mode, and **PAUSE** function, 100
- Fathoms, conversion to feet and meters, 38
- Feet, conversion:
  - to fathoms, 38
  - to meters, 35
- Feet per second, conversion to miles per hour, 41
- Financial statement analysis ratio, 66-67, 76-77
- Frequency distribution, 52-56
  - condensed from raw scores, 57
  - interval width of, 53-56
- From-To plot, 123-124
- Furlongs, conversion to miles and kilometers, 38-39
- Gallons, conversion to liters, 32
- GOSUB** command, 6
  - (*See also* Subroutines)
- GOTO** command, and **GOSUB**, compared, 6
- Grads, conversion to radians and degrees, 46
- Graphics mode, 3
- Graphing, 112-126
  - choices of methods of, 114-116
  - PLOT** vs. **PRINT** form, 112-113
- Grouped frequency distribution:
  - condensed from raw scores, 57-58
  - median, 59-60
  - percentile rank, 61-62
- Hex numbers, conversion to decimal:
  - large numbers, 49-50
  - small numbers, 48
- Histogram (*see* Bar graph)
- Horizontal bar graph, 114
  - plotted, 121
  - printed, 120
- I key, use of, 4
- IF . . . GOSUB** (conditional subroutine call), 9-10
- Inches, conversion to centimeters, 34
- Information group (*see* Information substring)
- Information strings (Storage string; String storage), 67-70
  - alphabetizing substrings in, 96
  - to construct a string table, 142-143
  - deleting a substring from, 81
  - input and output programs for, 70
  - inputting and printing on a ledger sheet, 138-139
  - inserting a substring in, 80-81
  - inserting a substring
    - alphabetically in, 81-83
  - retrieval from, 83
  - utility of, 67
- Information substring (of an information string)
  - (Information group), 67-70
  - format of, 67-68
  - printing of, 83-84
- INKEY\$** function:
  - advantages of, 99-100
  - limitations of, 100-101

- INKEY\$** subroutines, 100-102, 105-108
  - inputting a number, 105-106
  - inputting a space, 106-107
  - print and save input, 107-108
  - print input at cursor, 106
- Input variables (for subroutines in this book), modifying, 12-16
- Inputting:
  - to an array, 51-54
  - to an information string, 70
  - for ledger form subroutines, 130-131, 138-139
- Interest:
  - compound, 73, 74
  - simple, 70-72
- Interrelated data, vs. discontinuous data, 127-128
- Interval width of a frequency distribution, 53-56
- Inverse characters, 3
- Isosceles triangle, area of, 21
- Jumps (**GOTO**, **GOSUB**), 6-7
- Keyboarded characters, reported by **INKEY\$**, 99-100
- Keywords, **BASIC**:
  - abbreviated forms of, 4
  - boldface used for, 2
- Kilometers, conversion:
  - to light years, 41
  - to miles, 37
  - to nautical miles, 40
- Kilometers per hour, conversion:
  - to meters per second, 42
  - to miles per hour, 43
- Large numbers, conversion of, 30
- Last-name-first subroutine, 95-96
- Ledger sheet subroutines, 128, 136-140
  - displaying a ledger sheet, 136
  - input information string on and print, 139-140
  - input string array on and print, 137-138
  - print information string on, 138-139
  - print string array on, 136-137
- Light years, conversion to kilometers and miles, 40
- Line numbers:
  - for subroutines in this book, 10-12
  - variable, 10-11
- Linear measures, conversion of, 34-41
- Linear regression:
  - calculating a prediction line by, 64-65
  - in statistical subroutines, 55-56
- Liquid measures, conversion between English and metric, 31-34
- Liters, conversion:
  - to gallons, 32
  - to quarts, 32
- M key, 18
- Machine code programs:
  - caution in use of, 134-135
  - example of, 149-152
  - for scrolling, 134-135
  - USR** function to call, 134-135

- Magnitude, maximum and minimum, 91
- Matrix table, 131-135, 140-142
  - changing an array element of, 141-142
  - changing an information subgroup of, 143
  - changing a string-array element of, 144-145
  - constructed from an array, 140-141
  - constructed from an information string, 142-143
  - constructed from a string array, 144
  - (*see also* Super matrix)
- Maturity value:
  - at compound interest, 73-74
  - at simple interest, 70-71
- Maximum, 85
  - of an array, 89-91
  - magnitude (absolute value), 91
- Mean, of an array, 55-56, 58-59, 90
- Measures, conversion of, 29-50
- Median:
  - of an array, 55-56, 58-59
  - for grouped frequency distribution, 59-60
- Memory, conserving, 4, 8-9
  - (*See also* Efficiency)
- Meters, conversions:
  - to feet, 35
  - to various English measures, 37
  - to yards, 36
- Meters per second, conversion to kilometers per hour, 42
- Miles, conversion:
  - to furlongs, 39
  - to kilometers, 37
  - to light years, 41
  - to nautical miles, 40
- Miles per hour, conversion:
  - to feet per second, 41
  - to kilometers per hour, 42-43
- Milliliters, conversion to ounces, 31
- Minimum:
  - of an array, 89
  - magnitude (absolute value), 91
- Mode, of an array, 55-56, 58-59
- Moving average trend, 79-80
- Name routine (last name first), 95-96
- Nautical miles, conversion to miles and kilometers, 39
- Nested subroutines, 9
- Number bases, conversion between, 47-50
- Numbers:
  - conversion of, small vs. large, 30
  - sorting, 93-95
- Ounces, conversion to milliliters, 31
- Output variables (for subroutines in this book), modifying, 13-17
- Outputting, from information string, 70
- Partition sort, 94-95
- PAUSE** command, for fast and slow modes, 100

Payable note:

- bank discount of, 72
- true discount of, 73

Payments:

- at interest, 70-75
- monthly, on fixed payment, 74-75
- monthly, when number of payments known, 75

**PEEK** function, 103-104

Percentile rank:

- in grouped frequency distribution of scores, 61-62
- of single score, 60-61

Phonebook subroutine, example of information string, 68-70

**PI**, 18

Pie graph, 124-126

- displaying, 124
- segment plot, 125-126
- segment print, 126
- two-part, 125

Pixel, 112-113

**PLOT** statement:

- for color computers, 116-117
- in graphing, 112

Plotting:

- within cartesian coordinates, 121-122
- a cartesian line between two points, 123-124
- within polar coordinates, 122-123

Point graph:

- plotted, 118, 123-124
- printed, 117-118

**POKE** function, 103-104

Polar coordinates:

- displaying, 122
- plotting within, 122-123

Polygon, regular, area of, 23

Prediction line, by linear regression, 64-65

**PRINT** graphing subroutines, 112-113

- for color computers, 116-117

Programmers, novice and experienced, 1-2

Programming:

- ideas for, from subroutines in this book, 5
- subroutines as building blocks in, 1
- (See also BASIC programs)

Programs, user-friendly, 99-100

Quarts, conversion to liters, 31

Radians, conversion to degrees and grads, 46

Random numbers, 85-86, 92-93

- controlling range of, 92
- generating unique values of, 92-93

Raw scores:

- condense to frequency distribution, 57
- condense to grouped frequency distribution, 57-58

Rectangle, area of, 22-23

**RETURN** statement, placement in subroutine, 6-7

S-POSN system variable, 104-105

**SAVE** command, suggestions for use of, 4-5

- Savings:
  - future value of, 76
  - payment for certain value, 75-76
- Score, single, percentile rank of, 60-61
- SCR-CT variable, 104
- Screen, pixels on, 112
- SCROLL** function:
  - enter to **SCROLL**, 108-109
  - overcoming defects of, 102-105
- Scroll prompt, on color computers, 104
- Scrolling, 108-111
  - automatic, 108-109
  - on color computers, 133-134
  - with footing, 110-111
  - with heading, 109
  - machine code for, 149-152
  - sideways, 133-135
  - and user-friendly programs, 99-100
  - window of super-matrix, 146-152
- Search, 97-98
  - binary, 88, 97-98
  - sequential, 97
- Set of values (*see* Array)
- Simple interest:
  - calculation of, 70-72
  - and maturity value, 70-71
  - with monthly payment, 71-72
- Sinclair color computers (in general) (*see* Color computers; Sinclair Spectrum color computer; Timex/Sinclair 2000)
- Sinclair Spectrum color computer, 3-4, 99-100, 112-113, 116-117, 127-128
- Sinclair ZX80 and ZX81 computers, 99-100
- Sinking fund, 75-76
- Slow mode, 100
- Small number conversion, 30
- Solids, surface area of, 25-26
- Sorting, 85-87
  - bubble, 86, 93
  - bubble and save, 86, 93-94
  - partition, 94-95
- Space character:
  - input into **INKEY\$**, 106-107
  - underline used for, in this book, 2
- Sphere:
  - surface area of, 25
  - volume of, 27
- Square, area of, 21-22
- Square kilometers, conversion to acres, 43
- Square measures, conversion of, 43-45
- Square meters, conversion to square yards, 44
- Square yards, conversion:
  - to acres, 44
  - to square meters, 44
- Standard deviation, 62-63
- Statistics subroutines, 51-65
- STOP** command, placed
  - between main program and subroutines, 17
- Storage string (*see* Information string)
- Straight line depreciation method, 77
- String array, two-dimensional, 128-130
  - change element of, 144-145
  - to construct a matrix, 144
  - input into and print, 137-138
  - print on ledger, 136-137

- String matrix table (*see* Matrix table)
- String storage (*see* Information string)
- String table (*see* Matrix table)
- Subroutines (in general), 6-17
  - conditional calls of, using **IF**, 9-10
  - definition and function of, 6-11
  - nesting of, 9
  - one- and two-line (useless), 7-8
  - placed near beginning of program, 16-17
  - as program building blocks, 1
  - single vs. multiple use of, 8-9
  - as source of programming ideas, 5
  - variable, using variable line numbers, 10-11
- Subroutines (in this book):
  - arrangement of, 1-2
  - modifying, for your program, 10-19, 30, 51, 104, 127-128, 130-131
  - renaming variables in, 12-16
  - renumbering lines of, 10-12
- Substring (*see* Information substring)
- Sum-of-the-years-digits depreciation method, 77-78
- Super matrix, 131-135, 145-152
  - print a window of, 145
  - scrolling down, 146-148
  - shifting the window of, 146, 148-152
- Surface area:
  - cone, 26
  - cylinder, 25
  - sphere, 25
- Tables, subroutines for, 127-152
- Temperature, conversion of, 45
- Timex/Sinclair color computers (1000 and 2000):
  - scrolling on, 104
  - string-handling capabilities, 87-88
- Timex/Sinclair 1000 computer, 1, 99-100
- Timex/Sinclair 2000 computer, 1, 3-4, 99-100, 112-113, 116-117, 127-128
- Trap, subroutine, how to avoid, 86-88
- Trend (moving average), 66-67, 79-80
- Triangle, area of, 20-21
- Trigonometric functions, 18
- Two-dimensional string array (*see* String array)
- Upper real limit, in stastical subroutines, 54-55
- User-friendly programs, 99-100
- USR** function, 134-135
- Values, set of (*see* Array)
- Variables (for subroutines in this book), input, working, and output, 12-16
- Variance, of an array, 62-63
- Volume:
  - cone, 28
  - cube, 26
  - cylinder, 27
  - sphere, 27

Window, of super-matrix:

printing, 145

sideways scrolling of, 133–  
134

Working variable (for

subroutines in this book),

modifying, 13–17

*x-y* input, 56

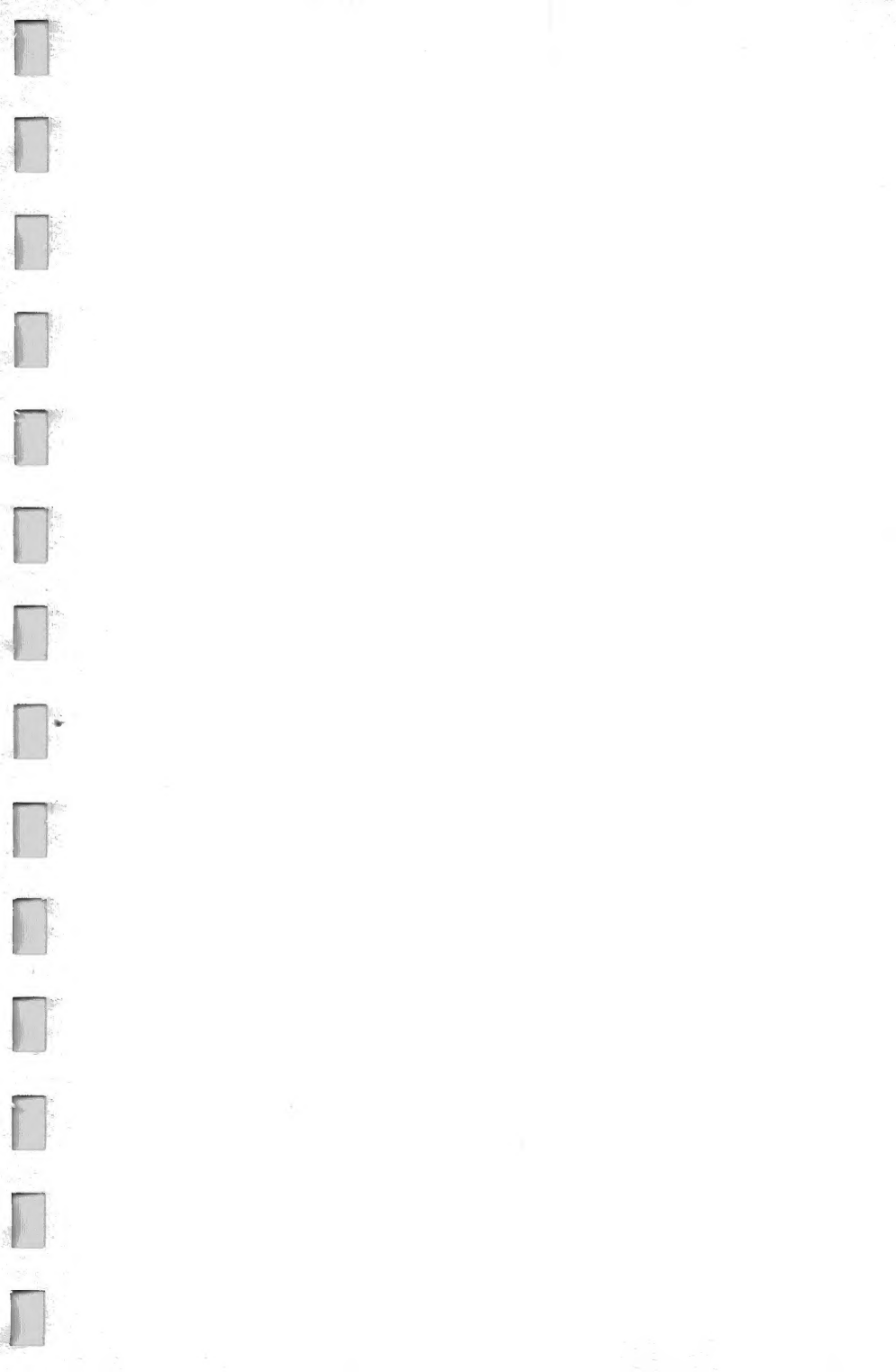
Yards, conversion to meters, 35

Z score, 55–56, 63–64

Z score equivalent, 63



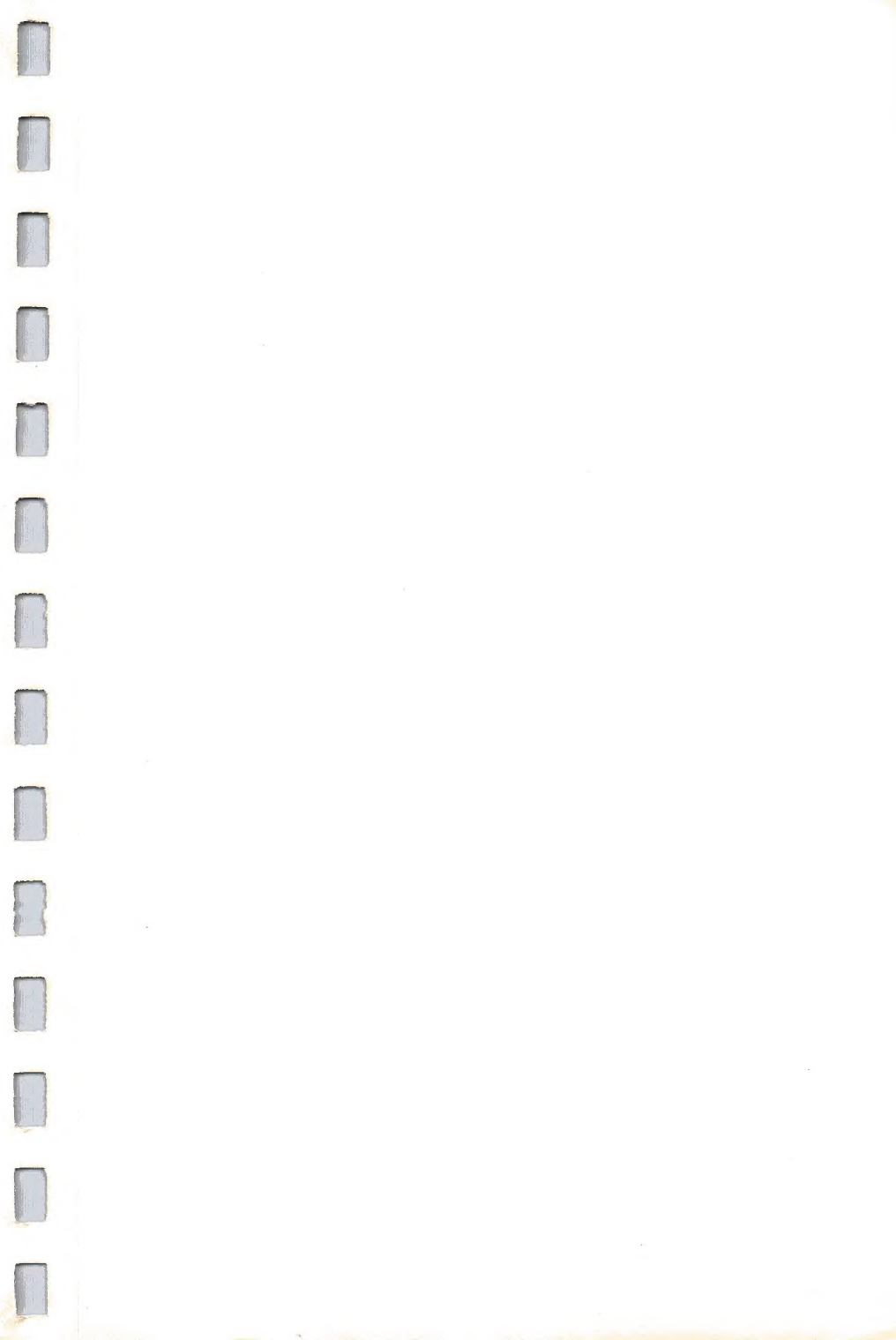




## Catalog

If you are interested in a list of fine Paperback books, covering a wide range of subjects and interests, send your name and address, requesting your free catalog, to:

McGraw-Hill Paperbacks  
1221 Avenue of Americas  
New York, N. Y. 10020



# GOSUBS

## 100 Program-Building Subroutines in Timex/Sinclair BASIC

**Ewin and Shirley Gaby**

Here in one book is a useful library of subroutines in Timex/Sinclair BASIC. The subroutines were written on the Timex/Sinclair 1000, but with minor modifications (clearly described in this book) can be used in programs written for the Timex/Sinclair 2000.

These subroutines have all been fully tested, and are ready to be put to work immediately as building-blocks within larger programs. The subroutines can be used to perform a wide variety of tasks, including conversions between different measurement systems, computations in statistics, geometry, and finance, and instructions for the storage and display of information.

Unusual and hard-to-follow routines are clearly explained. The book contains a general introduction to the subject for novices and, for those already familiar with the subject, suggestions about the best techniques for writing original subroutines.

Subroutines are among the most powerful tools available to a programmer, and for owners of Timex/Sinclair computers, this book is the ideal introduction.

**Ewin** and **Shirley Gaby** are the owners of S/E Gaby Associates, Inc., a business consulting firm specializing in communications and training. Mrs. Gaby is a graduate of Pennsylvania State University, where Mr. Gaby completed all course requirements toward a Ph.D. in Learning Systems. This is their first book.

